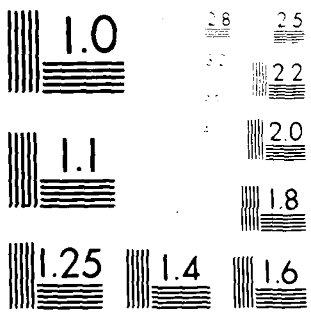
AD-A116 070 GEORGIA INST OF TECH ATLANTA ENGINEERING EXPERIMENT --ETC F/G 9/2
INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE--ETC(U)
MAR 82 L J GALLAHER F30602-78-C-0120

UNCLASSIFIED RADC-TR-82-46 NL

1 OF 2
AD A
116070


1.0
2.8
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(12)

AD A116070

RADC-TR-82-46
Final Technical Report
March 1982

INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE FOR DEVELOPING MACHINE INDEPENDENT SOFTWARE

Georgia Institute of Technology

L.J. Gallaher

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-82-46 has been reviewed and is approved for publication.

APPROVED: Elizabeth S. Kean

ELIZABETH S. KEAN
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-82-46	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE FOR DEVELOPING MACHINE INDEPENDENT SOFTWARE		5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report
		6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): L.J. Gallaher		8. CONTRACT OR GRANT NUMBER(s): F30602-78-C-0120
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Georgia Institute of Technology, Engineering Experiment Station, Atlanta GA 30332		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55811920
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (COES), Griffiss AFB NY 13441		12. REPORT DATE: March 1982
		13. NUMBER OF PAGES: 116
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same		15. SECURITY CLASS. (of this report): UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Elizabeth S. Kean (COES)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ada	TAN	machine independent
HOL	COTAN	Chebyshev
SIN	LOG	Polynomial
COS	MATH	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this investigation of the ability of Ada to support machine independent software, a library package of the elementary mathematical functions (sin, cos, en, etc.) was implemented and tested on the Ada/ED Compiler Version 11.4. The Ada language constructs proved quite useful and effective in creating the math function package. The programs were written and successfully syntax checked; however, flaws in this version of the compiler prevented a thorough debugging of these routines. (over)

DD FORM 1473, 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The routines were designed to be machine and accuracy independent. Accuracy independence was obtained using variable length polynomials whose coefficients are computed (at compile time) from Chebyshev series. For increased efficiency, the normally machine dependent operations ("bit picking") are isolated into subroutines that can be optimized for individual installations and hardware.

ABSTRACT

In this investigation of the ability of Ada to support machine independent software, a library package of the elementary mathematical functions (sin, cos, ln, etc.) was implemented and tested on the Ada/ED Compiler Version 11.4. The Ada language constructs proved quite useful and effective in creating the math function package. The programs were written and successfully syntax checked; however flaws in this version of the compiler prevented a thorough debugging of these routines.

The routines themselves were designed to be both machine and accuracy independent, that is the accuracy can be specified at the time the users integrate the package into their program. Accuracy independence was obtained using variable length polynomials whose coefficients are computed (at compile time) from Chebyshev series. For increased efficiency the normally machine dependent operations ("bit-picking") are isolated into subroutines that can be optimized for individual installations and hardware.

An appendix lists the library package.

X

A

TABLE OF CONTENTS

		Page
I.	Introduction	1
II.	Theory	2
	A. Range Reduction	4
	B. Chebyshev Approximation	7
	C. Other Approximations	9
III.	Ada Considerations	10
IV.	The Routines	11
	A. Introduction	11
	1. Variable Accuracy	11
	2. Questions of Efficiency	13
	3. Error Tolerances	18
	4. Error Conditions and Out-of-Range Alarms	22
	B. The Functions	[illegible]
	1. SIN and COS	[illegible]
	2. TAN	[illegible]
	3. ATAN	[illegible]
	4. ASIN	[illegible]
	5. ACOS	[illegible]
	6. EXP	30
	7. LN	[illegible]
	8. SQRT	[illegible]
	9. SINH and COSH	[illegible]
	10. TANH	[illegible]
	11. ASINH	[illegible]
	12. ACOSH	[illegible]
	13. ATANH	[illegible]
V.	Critique	38
	A. What Went Wrong	38
	B. What Went Right	[illegible]
	C. Unresolved Problems	[illegible]
VI.	Results and Conclusions	66
VII.	Recommendations for Further Study	[illegible]
	List of References	[illegible]
	Appendix A	
	Appendix B	

LIST OF TABLES

	Page
Table II.A-I	6
Table IV.A.1-I	13
Table IV.A.1-II	16
Table IV.A.2-I	17
Table IV.A.2-II	20
Table IV.A.4-I	25
Table IV.C-I	37
Table V.A-I	38

LIST OF FIGURES

	Page
Figure IV.A.4-I	26
Figure V.B-I	41
Figure V.B-II	42

1. **Introduction**

The goal of this project is to write a machine and accuracy independent library of elementary functions (sin, cos, exp, ... etc ...) in Ada, and have routines that would be reasonably close to being as efficient as could be written in assembly language and tailored to a particular machine.

In the past these elementary functions have been written in machine language and incorporated into the higher order language program primarily by being intrinsic to the language (as for example in Fortran and Algol) and were handled by the compiler. The accuracy is fixed at the accuracy of the machine for which compiler is written; if extended precision is to be available (as in Fortran) then a separate function is used.

In most cases the functions are first reduced into some reduced range and then approximated over that range by a Chebyshev polynomial, between power series. The coefficients of the approximation are precomputed; the desired accuracy determines the number of Chebyshev terms in the series. The Chebyshev polynomials themselves are not used but the sums of these coefficients of like powers of the variable are used as the constants in the power series. Since the Chebyshev polynomials contain all powers up to the order of the polynomial, adding one more Chebyshev term (to get more accuracy for example) changes all the terms in the power series. This is no problem if the power series coefficients are to be determined once and for all at the time the functions are coded, but for variable accuracy both the number and values of the coefficients of the power series will change.

The object of trying to implement these elementary functions with variable accuracy methods is to be aware of the computations. If some accuracy is acceptable, this should be achievable with reduced computations. For a

Chebyshev derived power series approximation, this will mean fewer terms in the power series. Since accuracy requirements are known at compile time, using generics, it is possible in Ada to determine both the number of Chebyshev terms and the values of the coefficients of the power series at that point.

In the case of the exponential, square root and logarithm functions, polynomial approximations are used after mapping the function onto some relatively small interval, associated with a power of 2. This kind of mapping is generally done by manipulating the exponent of the argument, a process most easily done in assembly or machine code. While this can be done in a machine-independent manner in Ada, it is questionable that it can be done efficiently. This suggests that while most of the code for EXP, SQRT and LN functions [illegible]

[illegible]

[illegible]

II. Theory

[illegible]

for the elementary math functions; during the late fifties and early sixties it held a unique position in almost every computing center library. As demand for greater accuracy increased, improved approximations began to appear in the literature until in 1968 "Approximations for Digital Computers" by Hart, et al. superseded Hastings as the standard reference. This monumental work brought together the best techniques then available and supplied tables of coefficients for several thousand different approximations. The work done here leans heavily on the work of Hart and company as the many references indicate; however this dependency is mainly in philosophy of approach. All of the actual (Chebyshev) coefficients used in these programs were calculated separately and not taken from Hart. This was made necessary by the fact that the form of the coefficients in Hart, as coefficients of the power series polynomials, is not the form needed here; the form needed here is as coefficients of the Chebyshev series. But since nearly all of the approximations here are equivalent to one of the other form given in Hart, we could check on and use the corresponding error estimates given there, and these are cited where they are expected to conform.

Hart and coauthors give a large number of approximations for the various elementary functions in the "Padé" form, as $P(x)/Q(x)$ where P and Q are polynomials. The approximations used here can be considered of the same form but with the restriction $Q(x) = 1$; that is, we used just simple single polynomial approximations, not the ratio of two polynomials. There were several reasons for this. First, it is not at all clear that the P/Q approximations are significantly better (i.e., faster for the same accuracy) than the corresponding single polynomial with the same number of total arithmetic operations. Much depends on the status of the divide operation -- whether it is fast or slow is the floating point divide operation relative to the multiply

operation. In 1960 it was assumed that the hardware speed ratio for multiply/divide was about 12/13. Today this ratio is more typically 1/2 to 1/4 with multiply gaining in speed relative to divide as newer models appear. In fact some of the array processors have no divide at all. Under these circumstances it seems highly questionable as to whether P(x)/Q(x) approximations are significantly faster than the simpler single polynomial.

Other reasons for restricting to a single polynomial is that it is then much simpler to find the near optimal polynomial for a given accuracy -- the Chebyshev approximations do this -- and the number of constants that need to be stored is then relatively small.

II. A **Range Reduction**

There are many ways to increase the efficiency in approximating a function and one of the more important of these is to reduce the range over which the approximation must hold. Use of properties such as periodicity, addition theorems and symmetry techniques are quite effective. Thus for example the trigonometric functions sine and cosine. By the 2π periodicity one can first reduce the range of the approximation to the interval ±π. Next one can reduce the interval to ±π/2 since

[illegible]

[illegible]

and finally one can reduce the range to ±π/4 by the relations

[illegible]

[illegible]

This then the sine and cosine interchange roles in the last transformation or when the sine is computed from the cosine approximation and the cosine from the sine approximation. A further reduction to the interval 0 to π/4 can be achieved by making use of symmetry. However the symmetry is usually exploited

by the use of even or odd polynomials, i.e.,

$$\sin(x) \approx x P_s(x^2)$$

$$\cos(x) \approx P_c(x^2)$$

where P_s and P_c are the sine and cosine polynomial approximations.

Table II.A-I gives the ranges over which the various functions are approximated by polynomials. The various range reductions relations used are listed in the program. (See Appendix A.)

In doing range reduction for the functions EXP, LN and SQRT, it is assumed that the computer hardware in use is basically binary. (This is consistent with both the proposed IEEE standards and MCF or Nebula specifications.) For each of the functions LN and SQRT the argument is separated into a characteristic and an exponent of 2, i.e.,

$$x = 2^J p$$

where J is integer and $0.5 \le p < 1.0$.

Then for SQRT

$$SQRT(x) = SQRT(2^J p) = 2^{J/2} \sqrt{p},$$

and $\sqrt{p}$ is approximated by a polynomial.

For LN

$$LN(x) = LN(2^J p) = J \ln 2 + \ln p,$$

and $\ln p$ is approximated by a polynomial.

In the case of the EXP function the argument is first mapped onto a power of 2 by

$$x' = x/\ln 2,$$

and x' broken down into an integer and remainder part $x' = k + p$ where k is integer and $-0.5 \le p \le 0.5$ so that

$$e^x = 2^{x/\ln 2} = 2^{x'} = 2^{k+p} = 2^k\, 2^p.$$

Here $2^{|p|}$ is approximated by a polynomial and if $p \le 0$ the reciprocal is

TABLE II.A-I

Ranges over which polynomial approximations are used for the various functions.

Function	Interval
SIN	$\pm \pi/4$,
COS	$\pm \pi/4$
TAN	$\pm \pi/8$
ATAN	$\pm(\sqrt{2} - 1)$
ASIN	± 0.375
EXP	0, 0.5
LN	0.5, 1.0
SQRT	0.5, 1.0
SINH	± 1.0
COSH	± 1.0
TANH	± 0.5
ASINH	± 0.375
ATANH	± 0.25
ACOS	polynomial not used
ACOSH	polynomial not used

taken. The operations dealing with the powers of 2 are assumed to be very fast; they will be machine dependent and should be tailored specifically for each machine.

The hyperbolic functions and their inverses are approximated by polynomials in a narrow region around the origin and outside this region are computed from their relationship to EXP, LN and SQRT.

The inverse functions arccos(x) and invcosh(x) have an anomalous behavior in the neighborhood of x = 1; they cannot be approximated by a polynomial in x near x = 1. Instead we have

$$\arccos(x) \approx \sqrt{2y + y^2/3 + 4y^3/45 + \ldots}$$

where y = 1-x, and

$$\text{invcosh}(x) \approx \sqrt{2y - y^2/3 + y^3/45 - \ldots}$$

where y = x-1.

The first two terms are used to approximate the functions,and the third term is used to control the range over which the approximation is used in such a way as to maintain the error tolerance. Outside this range the arccosine is computed from the arcsine; the invcosh is computed from LN and SQRT.

For the arcsine, an odd power series is used in the region about the origin; outside this region the arcsine is computed from its relation to arctangent and SQRT function, i.e.,

$$\text{arcsine}(x) = \arctan(x/\sqrt{1-x^2}).$$

II. B. Chebyshev Approximations

The advantage of the Chebyshev polynomials is that they are the polynomials having the largest number of maxima and minima on the given interval and for which all maxima and minima are equal in magnitude. Thus, if the lowest order term neglected can be considered the error function, it can be seen that the error is more or less uniformly distributed over the interval,

and can be shown to have the smallest maximum error of any polynomial approximation of the same order.

Since accuracy requirements are known at compile time, it is possible in Ada to determine both the number of Chebyshev terms and the values of the coefficients to the power series at that point.

Example: $\sin(\pi x)$.

This function can be mapped onto the interval $-1 \le x \le +1$. The function can then be well approximated in this range by the n-term Chebyshev series

$$\sin(\pi x) \equiv \sum_{1 \le k \le n} B_k T_k(x) \quad -1 \le x \le +1$$

where the Bs can be calculated from

$$B_k = \frac{2}{\pi} \int_{-1}^{1} T_k(x) \sin(\pi x) dx / \sqrt{1-x^2} .$$

(Only the odd numbered Bs will be nonzero.)

Each $T_k(x)$ is a polynomial of order k

$$T_k(x) = \sum_{o \le j \le k} C_{kj} x^j .$$

The Cs are well known and calculated exactly.

The n-term power series approximation for the sine then is

$$\sin(\pi x) \cong \sum_{1\le k\le n} \sum_{o\le j\le k} B_k C_{kj} x^j$$

$$\cong \sum_{o\le j\le n} A_j x^j$$

where

$$A_j = \sum_{j\le k\le n} B_k C_{kj} .$$

We see that adding more Chebyshev terms, say to improve the accuracy, changes all the A coefficients of the power series. Similar considerations apply to approximations for the other functions.

II.C. Other Approximations

In addition to Chebyshev derived polynomials, two other approximations commonly used are Pade' like approximations and continued fraction. The Padé like method consists of approximating as a function a ratio of two polynomials P(x)/Q(x); it can be converted to an equivalent continue fraction and vice versa. While these methods can sometimes be superior to a simple linear Chebyshev polynomial approximation in the sense that the same or better accuracy can be obtained with fewer computer operations, they all suffer the same drawbacks from the point of view of trying to write machine and accuracy independent programs. This is that for optimum conditions, all coefficients or constants in the method will change if the accuracy is changed. The Pade' and continued fraction methods suffer other problems.

First, it is much more difficult to find the optimum set of coefficients for a given accuracy, while this is relatively easy to do for the Chebyshev series. The Chebyshev series requires storing only a single set of constants

(10) for each function while for the P(x)/Q(x) method, an entire set of constants would be needed for each accuracy interval.

A second reason for not going with the ratio of polynomials is that it requires a division operation and the trend for hardware today is toward a relatively slow divide operation relative to addition and multiply. Present day hardware suggests we should avoid divisions where practical. (See IV.A.2 for a further discussion of this point.)

In summary, the Chebyshev method was chosen over the P(x)/Q(x), the ratio of two polynomials, because

1) Variable accuracy methods are relatively easier to obtain;
2) Fewer stored constants are required;
3) A division operation is eliminated (or traded for some number of multiplications and additions).

III. Ada Considerations

The principal Ada techniques expected to be most useful in developing machine and accuracy independent functions are the package and generic concepts. The main idea is that the functions will be embedded in a package with a generic parameter, call it REAL, describing the type (number of digits) to be used in the floating point arithmetic. The type REAL or its number of digits is then specified by the user in his main program. Then at the time the package and user program are integrated, the package is instantiated with the appropriate number of digits for REAL.

There are a number of program features that depend on the number of digits in REAL (called in Ada REAL'DIGITS).

First, the number of terms in the Chebyshev series and so the power series is determined by requiring that the Chebyshev terms not used are all smaller than 10**(-REAL'DIGITS). Next, this determines the size of the arrays that hold the power series polynomial coefficients. Then, in some cases

(ACOS, ACOSH, and TANH) the range in which a particular approximation is used will be determined by the number of digits in REAL; the computation of these ranges is carried out in the initialization block of the function package.

Certain assumptions concerning the computer arithmetic have been made in writing these packages. We believe these assumptions are, in general, consistent with the Ada view, the IEEE standards, and the MCF or Nebula choice of floating point arithmetic. These assumptions are as follow:

1. There is a maximum floating point number (called in Ada FLOAT'LARGE).
2. -FLOAT'LARGE exists and is the most negative floating point number.
3. Every floating point number including FLOAT'SMALL, but excepting zero, has a reciprocal; the reciprocal of ±FLOAT'LARGE may be zero, but need not be.
4. Any positive number can be subtracted from FLOAT'LARGE and any positive number can be added to -FLOAT'LARGE without causing an exception alarm.
5. FLOAT'LARGE can be divided by any number greater than or equal in magnitude to 1.0 or multiplied by any number less than or equal in magnitude to 1.0 without causing an exception.
6. Any two floating point numbers (including ±FLOAT'LARGE) can be compared without causing an overflow or exception.

IV. Routines

IV.A. Introduction

IV.A.1. Variable Accuracy

One of the prime goals of this effort was to construct these subroutines

so as to be able to give variable accuracy. The object is to let the user specify the accuracy to which the computations are to be carried out with the idea that the less accuracy required the less time needed for the computation; the user does not need to pay (timewise) for accuracy not needed. The variable accuracy is achieved here by using variable length polynomials for the approximations. The number of terms in the polynomial is determined at compilation time (or later at "assembly" time) by use of a generic parameter type REAL. REAL is a user specified type and it is the number of digits specified by the user for the type REAL that determines the accuracy and, consequently, the number of terms in the polynomial approximation for each function.

Once the accuracy is determined the polynomial coefficients are computed from the corresponding Chebyshev coefficients. These Chebyshev coefficients have been precalculated and are stored as part of the program or package containing the function. Enough terms in the Chebyshev series are stored to be able to achieve an accuracy (relative error) of 10^{-16}. It is assumed that this accuracy limitation (10^{-16}) will be adequate for the vast majority of the anticipated Ada applications in embedded systems; extension to higher accuracy would be straightforward. Table IV.A.1-I gives a table of the accuracy versus the number of terms needed for each function. There is, of course, for each function a certain amount of computation necessary to reduce the argument range to within the range of the polynomial, but that is not reflected in this table. (Also not reflected in Table IV.A.1-I is the fact that certain functions call other functions for some ranges of their values rather than using a polynomial.) Table IV.A.1-II lists those functions that call others and in what ranges.

TABLE II A.1-1

The order of the polynomial needed for each function is dependent on accuracy requirement

		ACCURACY			
Function	Interval	10^{-4}	10^{-8}	10^{-12}	10^{-16}
SIN	$\pm\pi/4$	2	3	5	6
COS	$\pm\pi/4$	2	4	5	6
TAN	$\pm\pi/8$	2	4	6	9
ATAN	$\pm(\sqrt{2} - 1)$	2	5	7	10
ASIN	± 0.375	2	4	7	9
EXP	0, 0.5	3	5	7	9
LN	0.5, 1.0	1	3	5	7
SQRT	0.5, 1.0	1	3	5	8
SINH	± 1.0	2	4	5	6
COSH	± 1.0	2	4	6	7
TANH	± 0.5	2	5	7	10
ATANH	± 0.25	2	4	6	8
ASINH	± 0.375	2	4	7	9
ACOS	polynomial not used				
ACOSH	polynomial not used				

Table [illegible]

Functions that call other functions and the ranges where this is done. These functions (except [illegible] and [illegible]) are [illegible] outside the ranges given here.

Function	Functions Called	Range
ASIN(X)	ATAN & SQRT	ABS(X) < 1 [illegible]
ACOS(X)	ATAN	[illegible]
"	SQRT	[illegible]
SINH(X)	EXP	[illegible]
COSH(X)	EXP	[illegible]
TANH(X)	EXP	[illegible]
ATANH(X)	LN	[illegible]
ASINH(X)	LN & SQRT	[illegible]
ACOSH(X)	LN & SQRT	[illegible]
"	SQRT	[illegible]

III.A.2 Questions of Efficiency

There are two kinds of efficiencies to be considered when writing library subroutines: space and time considerations. The ideal library routine is to run as fast as possible and take up as little space as possible. To a certain extent, these two considerations, space and time, can be traded off against each other. Faster routines can be written at the cost of taking up more room in storage, or shorter more compact routines can be written that will not run quite as fast. In the construction of the elementary function routines given here the emphasis has been placed on speed of running. While every effort was made to make these routines take up a small amount of space, the main emphasis has been on speed of execution.

There is an important feature of [illegible] that [illegible] a [illegible] tradeoff for time and space efficiencies. This is the [illegible] that [illegible] certain procedures or functions to be written [illegible] with [illegible] and thereby, avoid the time necessary for a procedure call. This [illegible] very much in [illegible] programs and [illegible] in [illegible] to [illegible] very [illegible] of functions which are then [illegible] frequently, and still have efficient (timewise) programs. Some programs are not yet [illegible] and the [illegible] program would only be [illegible] for a [illegible] [illegible]. [illegible] effort was made here to [illegible] this [illegible] of [illegible] to put comments in the various functions to indicate that [illegible] will be [illegible] in this manner.

In considering time efficiency, the important feature is the relative amount of time consumed by each of the computer operations. Since the functions here are intended to be used with floating point arithmetic, the assumption here is that only the floating point operations of addition (subtraction), multiplication and division need be considered in computing execution

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Table IV.A.2-1

In calculating the average computation time for the various functions, it is assumed that either the values of the input variable or the output values are uniformly distributed over some interval. This table gives this interval for the various functions and indicates whether it is the input or output interval assumed uniform.

Function	Input	Output	Interval
SIN	X		[illegible]
COS	X		[illegible]
TAN	X		[illegible]
ARCSIN		X	[illegible]
ARCCOS		X	[illegible]
ARCTAN		X	[illegible]
EXP	X		[illegible]
LN	X		[illegible]
SQRT	X		[illegible]

[illegible]

[illegible]

[illegible]

example, Floating Point Systems' AP-120B, a very high speed auxiliary array processor, doesn't even have a divide; addition and multiplication have the same speed, giving ratios of 1:1:∞ for its arithmetic speeds. Again this means that algorithms involving division will be at a significant disadvantage.

In any event, this suggests that one needs to make some kind of assumption concerning relative speeds of arithmetic operations in order to compare different algorithms. As a first approximation the speed ratios in Table IV.A.2.ii are used in comparing the efficiency of algorithms.

It is also clear from the observation of how slow divide is relative to multiply that one should be looking for algorithms that use as few divides as possible. This is one of the justifications for avoiding the P(x)/Q(x), the ratio of two polynomials, algorithms and using the simpler Chebyshev polynomials instead.

IV.A.3 Error Tolerance

One of the main goals here is to achieve an error tolerance consistent with the accuracy specified by the user. The user specifies the number of digits to be used for his particular type of floating point numbers or reals and this then determines the acceptable error. Thus for example, if the user specifies:

type REAL is digits 5;

the acceptable (relative) error tolerance will be taken as 10^{-5}. This implies then that the power series approximation will be composed of all Chebyshev terms whose absolute value is greater than or equal to 10^{-5}. This computation of the power series coefficients from the Chebyshev coefficients is done in the package initialization. It is also at this point that the size of the power series arrays is determined.

Table II.A.[illegible]-1

Assumed Relative Speeds for the Arithmetic Operations

Operation	Relative Speed
Add two real numbers	[illegible]
Multiply two reals or integers	[illegible]
Divide two reals or integers	[illegible]
Integer add or compare	Negligible
Logical operations	[illegible]
Multiplication and division by 2	[illegible]
Compare two real numbers	[illegible]

There is one exception to the above description of how the error is controlled and that is for the SQRT function. In the square root function a single Newton-Raphson or Heron's iteration is performed; i.e.,

$$x \leftarrow (y/x + x)/2$$

where $x = \sqrt{y}$. This operation squares the relative error when the error is small. For SQRT it is only the first approximation that is obtained from the Chebyshev polynomials, the final value is obtained from the above iteration. Thus the accuracy of the power series need be only as good as the square root of the relative error tolerance; i.e., if the relative error tolerance is 10^{-5}, the polynomial need only be as accurate as $10^{-2.5} = 3.16^{-5}$ -- the Heron iteration will improve this back to 10^{-5}.

The Chebyshev approximations normally give absolute not relative error limits. When dealing with floating point arithmetic it is relative error that one wishes to maintain more or less uniform over some internal. For the even functions this is not a problem since in the interval of approximation the functions do not change greatly and so the relative error is very nearly equal to the absolute error (the functions being approximately 1 at the origin). To maintain the relative error more or less constant for the odd functions, say $f(x)$, we use a Chebyshev approximation to $f(x)/x$. This gives nearly uniform relative error for $f(x)$ on the interval around $x = 0$, and again the absolute error is approximately equal to the relative since for these odd functions $\lim_{(x \to 0)} f(x)/x = 1$. So for example, the Chebyshev coefficients for the SIN function are actually those for $\sin(x)/x$, and those for TAN are in fact for $\tan(x)/x$, etc., for all the odd functions.

IV.A.4. Error Conditions and Out of Range Alarms

In nearly all computer arithmetic systems there is a limit to the size of a floating point number; there is also a smallest-in-magnitude nonzero number. These numbers are machine dependent but referable in Ada as F'LARGE and F'SMALL, where F refers to the (floating point) type. Because of these limits there are certain functions such as EXP, SINH, COSH, etc. for which only a relatively restricted range of inputs are valid. Thus for example, if we refer to our floating point type as REAL, input values to the EXP function greater than ln(REAL'LARGE) cannot be computed since they would result in values greater than REAL'LARGE. There are other functions such as LN and SQRT that have invalid ranges because the result would not be in the real number system (but would be complex). For example, numbers less than or equal to zero are invalid inputs for LN; $X < 0$ is invalid for SQRT(X).

The problem of what to do about invalid inputs is complicated by the fact that all computer arithmetic with floating point numbers involves rounding; floating point arithmetic is seldom exact. When rounding takes place in the neighborhood of the boundary between the valid and invalid region of the function, it is difficult to decide what is the best action to take. Thus for example, if one wants to evaluate:

```
SQRT(A + B + C - SIN(D))
```

and the exact value of A+B+C-SIN(D) is 0 but due to rounding (or truncation in SIN) the computer results turns out to be -1.0 E-9, what is the proper action? How should the SQRT subroutine respond to small negative inputs, and what is small? One action would be to trip a numeric exception alarm and let (force) the user handle the problem. Some systems halt the computation at this point.

Another solution is to assume that there is a small negative region such that input from that region ought really be treated as input of zero and not trip the exception alarm for these inputs, simply return zero; however it is difficult to determine how large or small this boundary interval should be.

The solution to the problem of the invalid input here is different yet from those mentioned above. We take the point of view that there should be no truly invalid input region and redefine our functions so all inputs are legitimate and will return some value and no exceptions are ever raised. For example, SQRT is defined so that

$$\mathrm{SQRT}(X) = \begin{cases} \sqrt{X} & X \geq 0, \\ 0 & X < 0 \end{cases}$$

The square root function given here always returns a value, with no exception raised for input values from -REAL'LARGE to REAL'LARGE. Thus, small rounding errors in the vicinity of zero will cause no difficulties; however, it is the users responsibility to trap any input values to SQRT that the user feels ought to be considered illegitimate. Table IV.A.4-I gives the extended definition of the elementary functions showing how the usually considered invalid regions are treated. As an example, Figure IV.A.4-I shows how the arcsine function is extended beyond the normal input range of $\pm$ 1.0.

As mentioned above the main virtue of extending the definitions of the functions into their normally invalid regions is to avoid extraneous error signals when the argument drifts into the invalid region by a small amount of rounding or truncation error. This solution may not be the best resolution to the problem nor the solution desired by a particular user. It does however have the virtue of allowing the user to determine what he would like to see as

Figure IV.A.4-I
The extended definition
of the arcsine function.

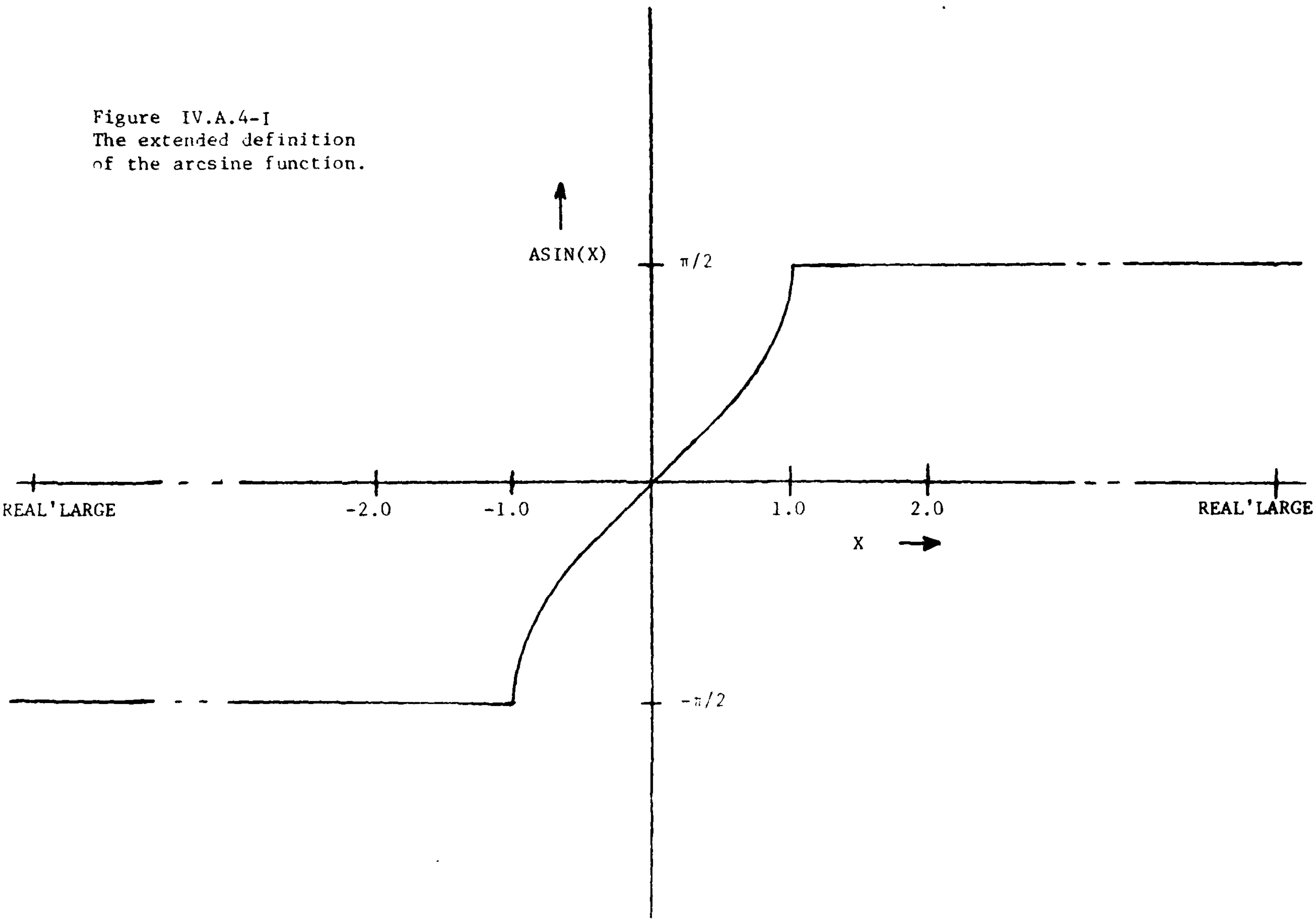


Table IV.A.4-I

A list of the functions and the way in which their inputs were extended to the normaly invalid regions, together with the ouput values in these regions.

SIN(x)	= sin(x)	all x
COS(x)	= cos(x)	all x
TAN(x)	= tan(x)	all x (returns $\pm$ REAL'LARGE for $x = \pm \pi/2, \pm 3\pi/2, \pm 5\pi/2 \ldots$)
ASIN(x)	= arcsin (x)	$\|x\| \le 1.0$
	$+\pi/2$	$x > 1.0$
	$-\pi/2$	$x < -1.0$
ACOS(x)	= arcos(x)	$\|x\| \le 1.0$
	0	$x > 1.0$
	$-\pi$	$x < -1.0$
ATAN(x)	= arctan(x)	all x
EXP(x)	= e^x	$\|x\| \le$ ln (REAL'LARGE)
	0	$x <$ -ln (REAL'LARGE)
	REAL'LARGE	$x >$ ln (REAL'LARGE)
LN(x)	= ln(x)	$x > 0$
	-REAL'LARGE	$x \le 0$
SQRT(x)	= $\sqrt{x}$	$x \ge 0$
	0	$x < 0$
SINH(x)	= sinh(x)	$\|x\| \le$ ln(REAL'LARGE)
	REAL'LARGE/2	$x >$ ln(REAL'LARGE)
	-REAL'LARGE/2	$x <$ -ln (REAL'LARGE)

Table IV.A.4-I
(Cont'd)

TANH(x) =	tanh(x)	all x
ASINH(x) =	invsinh(x)	all x
ACOSH(x) =	invcosh(x)	x >1.0
	0	x < 1.0
ATANH(x) =	invtanh(x)	|x| < 1.0
	REAL'LARGE	x > 1.0
	-REAL'LARGE	x < -1.0

the valid regions and allows (requires) him to set his own error traps for what that user considers illegitimate inputs.

IV.B.1. SIN and COS

Input Value Range: -REAL'LARGE to REAL'LARGE (in radians)

Output Value Range: -1.0 to +1.0

The argument, x, is first mapped onto the interval $-3\pi/2$ to $3\pi/2$. If $x \le \pi/4$ the sine or cosine polynomial is computed for the respective function; if $|x|$ is greater than $\pi/4$, SIN calls the cosine polynomial and COS calls the sine polynomial with the appropriate sign and phase adjustment.

There are no invalid input or output ranges for SIN and COS. However because of truncation and/or rounding errors it is possible for results to be returned which are slightly larger in magnitude than 1.0.

IV.B.2. TAN

Input Value Range: -REAL'LARGE to REAL'LARGE (in radians)

Output Range: -REAL'LARGE to REAL'LARGE

The argument, x, is first mapped onto the interval $-5\pi/8$ to $5\pi/8$. If $|x|$ then is less than $\pi/8$ the tangent polynomial is called; if $|x|$ is in the range $\pi/8$ to $3\pi/8$ the tangent polynomial for $|x| - \pi/4$ is computed (as Q) and TAN is evaluated as $(Q+1)/(1-Q)$, with appropriate sign[+]. If $|x| > 3\pi/8$, TAN is computed from the reciprocal of the tangent polynomial with appropriate sign and phase adjustment.

The only invalid input arguments are $|x| = m\pi/2$, $m=1, 3, 5$, etc. For these angles the value $\pm$ REAL'LARGE is returned. No error halt or exceptions are generated.

[+] This is an application of the relation :

$$\tan x = (\tan(x-b) + \tan b)/(1 - \tan(x-b))$$

with $b = \pm\pi/4$

IV.B.3. <u>ATAN</u>

Input Value Range: -REAL'LARGE to REAL'LARGE

Output Range: $-\pi/2$ to $\pi/2$ (in radians)

If the argument, x, is in the range $|x| \le \sqrt{2} - 1.0$, the arctan polynomial approximation is used. If $|x| > \sqrt{2} + 1.0$, then arctangent polynomial is used with $1/x$ as argument with appropriate sign and phase adjustment. In the intermediate range, that is $(\sqrt{2} - 1.0 < |x| \le \sqrt{2} + 1.0)$ the arctan polynomial is used with argument $(|x| - 1.0)/(|x| + 1.0)$ and appropriate phase and sign adjustment.

There is no invalid input range.

IV.B.4. <u>ASIN</u>

Valid Input Value Range: -1.0 to $+1.0$

Output Range: $-\pi/2$ to $\pi/2$ (in radians)

If the argument, x, is in the region $|x| \le 0.575$ then the arcsine polynomial approximation is used; if x lies outside this region then the relation:

$$\text{arcsine } x = \text{arctan}\left(x / \sqrt{1.0 - x^2}\right)$$

is used. Thus the arcsine routine uses both the arctan and the square root routine.

If the input is outside the valid region, that is outside -1.0 to 1.0, the value $\pm\pi/2$ is returned depending on the sign of the argument. Note: no error halt nor exception is generated for input values outside the valid region -1.0 to 1.0 (see Section IV.A.4 above).

IV.B.5. ACOS

Valid Input Value Range: -1.0 to +1.0

Output Range: 0.0 to π in radians

If the argument, x, is in the neighborhood of 1.0 then a special approximation is used as follows:

$$\theta = \sqrt{2y + y^2/3 + 4y^3/45 + \ldots}$$

where θ is the arccosine of x and $y = 1.0 - x$. By using the first two terms in this power series for small y the error in θ is approximately half the third term. The relative error is then approximately $\frac{4y^3}{90\theta_s} \approx \left(\frac{2y}{45}\right)^{5/2}$ where $\theta \approx \sqrt{2y}$ has been used. This relative error needs to be held below 10**(-REAL'DIGITS), so we need

$$\left(\frac{2y}{45}\right)^{5/2} \leq 10^{**}(\text{-REAL'DIGITS})$$

The range of y then is roughly $y < 4.0 \cdot 2.5$**(-REAL'DIGITS) (here $2.5 \approx 10^{2/5}$) and $4.0 \approx \sqrt[5]{45^2/2}$.

Outside the above range the arccosine is calculated from $\pi/2$ - arcsine. If the argument is outside the valid region, that is outside -1.0 to 1.0, the value 0 or π is returned depending on the sign of the argument. Note: no error halt nor exception is generated for input values outside the valid range (see Section IV.A.4 above).

[illegible]

Valid Input Value Range [illegible]

Output Range [illegible]

The [illegible]

[illegible]

[illegible]

Valid Input Value Range [illegible]

Output Range [illegible]

[illegible]

where [illegible] integer and [illegible] The natural log [illegible]

computed from the [illegible] polynomial in [illegible] and for [illegible] is [illegible].

If the argument, x, is outside REAL'SMALL to REAL'LARGE [illegible] then the value REAL'LARGE is returned, and an error trace and exception is generated; see Section IV.B.4 above.

IV.B.8. SQRT

Valid Input Value Range: 0 to REAL'LARGE

Output Range: 0 to REAL'LARGE

To compute the square root, the argument, x, is first decomposed into a binary exponent and a fraction part such that

$x = f\,2^{J}$

where J is integer and [illegible]. The square root of f is approximated [illegible] polynomial, and the square root of 2^{J} is obtained as $2^{J/2}$. The product of these two then is improved by the application of Newton's rule, which assures the relative error [illegible] within tolerance.

If the argument, x, is outside the valid range, that is outside 0 to REAL'LARGE (i.e., x < 0), then the value 0 is returned and an error trace and exception is generated. Note that because of rounding and/or truncation errors SQRT(x)**2 will not usually equal x and in particular SQRT(x)**2 may exceed REAL'LARGE even if x < REAL'LARGE.

Also, as a result of the Newton iteration, the error in SQRT is always positive.

IV.B.9. SINH and COSH

Valid Input Value Range: -ln(REAL'LARGE) to ln(REAL'LARGE)

Output Range: -REAL'LARGE/2 to REAL'LARGE/2

If the input argument, ϕ, is in the interval [illegible] a polynomial approximation is used [illegible]. If the argument is outside the interval [illegible], the functions are computed through their definitions in terms of the exponential, i.e.

$$\sinh \phi = (e^{\phi} - e^{-\phi})/2$$

$$\cosh \phi = (e^{\phi} + e^{-\phi})/2$$

If the argument, ϕ, is outside the [illegible] range the values [illegible] NEG.LARGE [illegible] are returned for the SINH and NEG.LARGE [illegible] for COSH. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] TANH

Argument range: [illegible] NEG.LARGE to NEG.LARGE

Function range: [illegible]

If the argument, ϕ, is in the interval [illegible] a polynomial approximation is used. For [illegible] [illegible] [illegible] [illegible] For ϕ in the interval [illegible] [illegible] the definition in terms of the exponential is used:

$$\tanh \phi = \frac{e^{\phi} - e^{-\phi}}{e^{\phi} + e^{-\phi}} = \text{sign}\,\phi\,\frac{1 - e^{-2|\phi|}}{1 + e^{-2|\phi|}}$$

Note that all inputs of x are valid. No error halts or exceptions are generated by TANH.

[illegible] ASINH

Valid Input Value Range: -REAL_LARGE to REAL_LARGE

Output Range: -ln(2*REAL_LARGE) to ln(2*REAL_LARGE)

If the input argument, x, is in the interval -0.375 to 0.375 then the hyperbolic sine is approximated with a polynomial. If $0.375 < |x| < \sqrt{REAL\_LARGE/2}$ then the ASINH is computed from

$$\ln(|x| + \sqrt{1 + x^2})$$

with appropriate sign. If $|x| > \sqrt{REAL\_LARGE/2}$, ASINH is computed from $\ln(2|x|)$ with appropriate sign.

Note that all inputs are valid and will return valid outputs. However, there will be values of x ($|x| > REAL\_LARGE/2$) for which

SINH(ASINH(x)) ≠ x and the discrepancy will be quite large. Also the relationship

ASINH(SINH(y)) = y should fail only for y values such that

$|y| > \ln(2 \cdot REAL\_LARGE)$.

[illegible] ACOSH

Valid Input Value Range: 1.0 to REAL_LARGE

Output Range: 0 to ln(2*REAL_LARGE)

If the argument, x, is in the neighborhood of 1.0, then a special approximation is used, as follows:

$$a = \sqrt{2y - y^2/3 + 4y^3/45 - \ldots}$$

where θ is the inverse hyperbolic cosine and $y = x - 1.0$. By using the first two terms in this power series, for small y the error in θ is approximately half the third term; the relative error is then approximately

$$\frac{4y^3}{2045} \approx \frac{\sqrt{2}\,y^{3/2}}{45}$$

where $\theta \approx \sqrt{2y}$ has been used. This relative error needs to be held below 1.0E(-REAL'DIGITS), so we need

$$\frac{\sqrt{2}\,y^{3/2}}{45} < 1.0E(-REAL'DIGITS)$$

or the range of y then is roughly

$y <$ [illegible] E(-2/3 REAL'DIGITS), here [illegible] $\approx 10^{2/3}$ and [illegible] $\approx \sqrt[3]{45^2/2}$.

Outside of the above range the approximation is divided into two regions. For $x < \sqrt{REAL'LARGE}/2$ the approximation $\ln(\sqrt{x^2-1} + x)$ is used and above this $\ln(2x)$ is used for ACOSH.

For invalid inputs, that is for $x < 1.0$, the value zero is returned; no error halt or exception is generated (see Section IV.A.4 above).

There will be values of x ($x >$ REAL'LARGE/2) for which the relation

COSH(ACOSH(x)) = x will fail and the discrepancy will be quite large. Also the relation

ACOSH(COSH(y)) = y

will fail badly for y values such that

$y >$ ln(REAL'LARGE/2).

IV.B.13 ATANH

Valid Input Value Range: -1.0 to 1.0

Output range: -REAL'LARGE to REAL'LARGE

If the argument, x, is in the interval -0.25 to 0.25 a polynomial approximation is used for ATANH. In the interval $0.25 < |x| < 1.0$ the relation $\ln((1 + x)/(1-x))/2$ is used.

For the invalid input region $|x| > 1.0$, the value $\pm$ REAL'LARGE is returned and no error halt or exception is raised; this point is discussed in Section IV.A.4.

IV.C. Verification of the Function Package

Short of proving that each function given here is correct, the best that can be done is to check the correctness of the value computed for a large number of arguments for each function. Since at this point we have no way of obtaining check values internally in Ada, it appears that to check them, we must write them out and verify each value by hand or by another computer program in a different language such as FORTRAN.

There is, however, another good method of verifying the correctness of a large number of values of a function and that is by checking certain addition theorems for these functions. For example the sine and cosine should obey the relation $\sin^2 x + \cos^2 x = 1$ for all x. This, however, is not a very good check since the relative error of the smaller of these could be very large but not be observed. A better check are the "triple relations". For example

$$\sin 3x = 3 \sin x - 4 \sin^3 x$$

can be used to verify the sine function. The triple relations have the advantage of requiring just one of the functions at a time and allows it to be

checked against itself. Table IV.C-I gives a set of triple relations for the functions considered here. A very good verification procedure would be to check the triple relation of each function for several thousand (random) points uniformly distributed over a range that would ensure the exercising of all branches of the function code. Let us emphasize again that this has not been done yet due to lack of time and not having a suitable Ada compiler. Only the exponential function has been executed and partially verified; Figures V.B-I and II give triple relation error curves of EXP for a limited sample of arguments.

V. Critique

V.A. What Went Wrong

The most serious difficulty encountered by this project was the lack of an adequate Ada compiler. In fact not until 30 days before the final termination date of the project (60 days after the original termination date, a three month no cost extension was requested and granted), did we obtain the Ada/ED compiler. Ada/ED (Version 11.4) was obtained through NTIS and is considered a preliminary unvalidated version intended for education and experimental use only. And while it was extremely complete, it did have a variety of bugs or errors; those we uncovered are listed in Table V.A-I. These errors, while not serious once they were understood, contributed significantly to our confusion. We were, in fact, learning about Ada and the difficulties of untangling our mistakes and misapprehensions about Ada from errors in Version 11.4 proved a real strain and lead to the consumption of vast amounts of computer time. (Ada/ED is an experimental version and not designed for production use -- it is very slow -- by a factor of about 10^4 to

Table IV.C-I

The "Triple Relations" for Verifying the Accuracy of the Elementary Functions.

$\sin 3x = 3 \sin x - 4 \sin^3 x$

$\cos 3x = -3 \cos x + 4 \cos^3 x$

$\tan 3x = (3 \tan x - \tan^3 x)/(1 - 3 \tan^2 x)$

$e^{3x} = (e^x)^3$

$\ln 3x = \ln 3 + \ln x$ or $\ln x^3 = 3 \ln x$

$\sqrt{3x} = \sqrt{3}\sqrt{x}$ or $\sqrt{x^3} = (\sqrt{x}\ )^3$

$\sinh 3x = 3 \sinh x + 4 \sinh^3 x$

$\cosh 3x = -3 \cosh x + 4 \cosh^3 x$

$\tanh 3x = (3 \tanh x + \tanh^3 x)/(1 + 3 \tanh^2 x)$

$3 \sin^{-1} x = \sin^{-1}(3x - 4x^3)$

$3 \cos^{-1} x = \cos^{-1}(-3x + 4x^3)$

$3 \tan^{-1} x = \tan^{-1}((3x - x^3)/(1-3 x^2))$

$3 \sinh^{-1} x = \sinh^{-1}(3x + 4x^3)$

$3 \cosh^{-1} x = \cosh^{-1}(-3x + 4x^3)$

$3 \tanh^{-1} x = \tanh^{-1}((3x + x^3)/(1 + 3x^2))$

Table V.A-I

A list of known flaws (as of 10/26/81) in the Ada/ED Compiler Version 11.4

Compile Time:

1. Generic and actual parameters may not have the same name.
2. LONG_FLOAT not implemented.
3. Won't handle exponentiation of a universal constant.

Run Time:

1. Won't multiply by a floating point number if its value is 0.0.
2. INTEGER truncates instead of rounds (sometimes?).
3. Universal constants as parameters of functions or procedures cause a RUN time error halt, not a compile time error.

10^5 (in execution) over what will be required in a production mode.)

The net result of not obtaining a compiler until so late in the program and then to have it be somewhat flawed has been that the functions and packages here are not very well debugged. In fact we will have to include the disclaimer that "this package of functions is an unvalidated early version and is intended for experimental use only". We would like to be able to claim at this point that each of the functions has been thoroughly debugged and can be certified correct; but this is not at all the case. The best we can claim is that they have been syntax checked; except for the EXP function, none of the functions given here have been executed. Preliminary debugging of the EXP function has been accomplished and error curves obtained showing that EXP does meet the error tolerances requested; however, even here more checking should be done before it is released for incorporation into a "live" system.

V.B. What Went Right

The main bright note of this project is that most of the ideas about how to use Ada to build library packages seem to have checked out. To summarize these briefly:

1) Packages; the packaging feature of Ada works nicely and is just what is needed for writing library subroutines.

2) Generics; also work well and provide a mechanism for constructing variable accuracy routines where the error tolerance is effectively supplied by the user at the time the package is integrated into his total program. This has been demonstrated explicitly for the EXP function where the error curves show how the accuracy depends on the users specification of the number of digits in his REAL type variables; see Figures V.B-I and II.

3) Overloading; this was a useful and convenient device but was not really crucial to setting up the library routines. The main advantage of overloading was that it allowed mixed mode arithmetic (most of the arithmetic operators were overloaded for various combinations of INTEGER, REAL, FLOAT and LONG_FLOAT).

4) Function and procedures; these constructs also worked well and of course were indispensable in setting up an elementary math function library.

5) Dynamic Arrays; worked well and were especially useful for library routines in allocating exactly the needed storage (for the polynomial coefficients for example).

6) Isolation of machine dependent operation; most of the operations expected to be machine dependent were isolated into small, short functions. These are mainly the bit-picking operations and are to be tailored by each installation to their particular hardware. By using the pragma INLINE, these operations can be made run time efficient. However, since the INLINE pragma is not implemented in the Ada/ED version 11.4, it was not possible to demonstrate explicitly here that this works well. We were able to demonstrate that this is not a difficult or unnatural way to proceed.

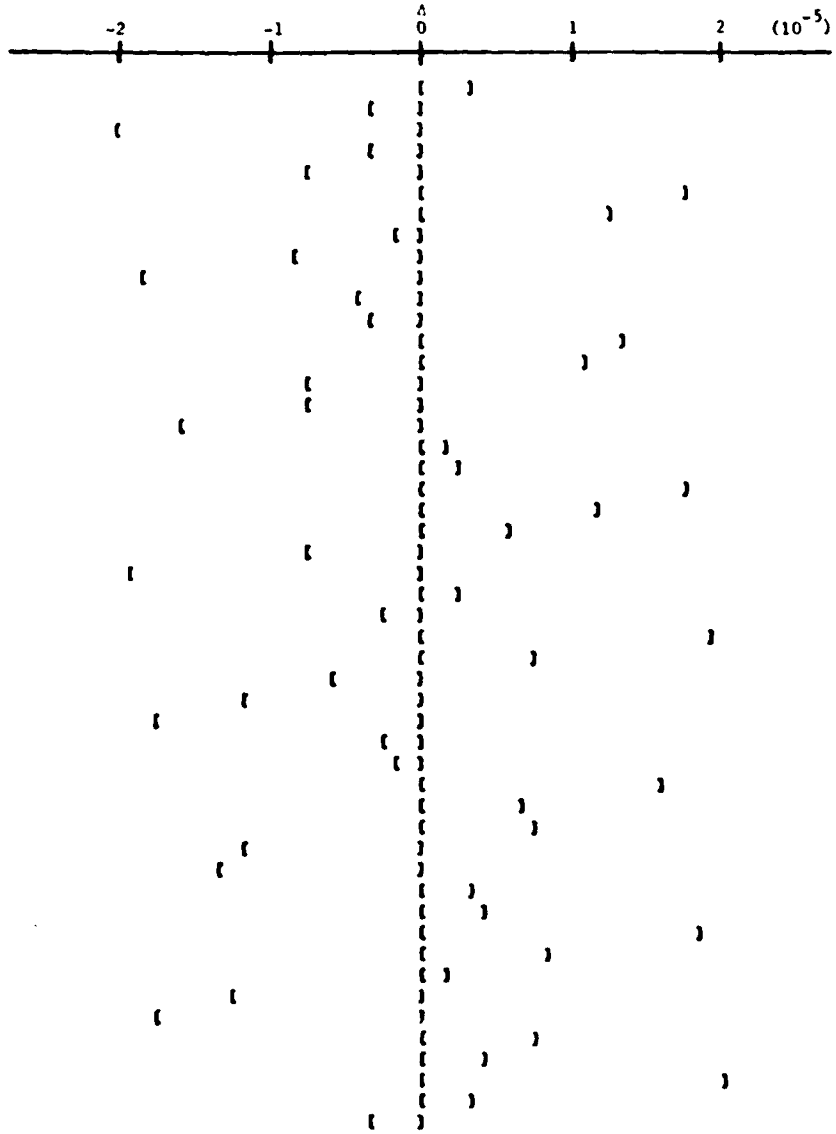
As mentioned earlier we were able to demonstrate that the elementary function package is syntactically correct and have been able to execute a reduced package containing just the EXP function. Figures V.B-I and II show error curves for the EXP function. (These error curves were generated by comparing e^{3x} with $(e^x)^3$ for 50 values of x. Note that $\Delta = e^{3x}/(e^x)^3 - 1$ can be up to four times the error tolerance since cubing e^x will triple the relative error and the error in e^{3x} can be in the opposite direction.) It would be desirable to run several more such curves at different (higher) accuracy requests before certifying the correctness of the EXP

Figure V.B-I

An error curve for the EXP function.

The measure of the relative error, Δ = EXP(3X)/EXP(X)**3 - 1, is plotted here vs X. REAL'DIGITS = 5 here giving an error tolerance of 10^{-5}.

3X	$\|\Delta\|$
2.4500E+00	3.5763E-06
2.3500E+00	3.1590E-06
2.2500E+00	2.0027E-05
2.1500E+00	3.4571E-06
2.0500E+00	7.2718E-06
1.9500E+00	1.7524E-05
1.8500E+00	1.2398E-05
1.7500E+00	1.8477E-06
1.6500E+00	8.8811E-06
1.5500E+00	1.8835E-05
1.4500E+00	4.4703E-06
1.3500E+00	3.6955E-06
1.2500E+00	1.3828E-05
1.1500E+00	1.1206E-05
1.0500E+00	7.6890E-06
9.5000E-01	7.3910E-06
8.5000E-01	1.6093E-05
7.5000E-01	2.0266E-06
6.5000E-01	2.2650E-06
5.5000E-01	1.7405E-05
4.5000E-01	1.2159E-05
3.5000E-01	6.3181E-06
2.5000E-01	7.8082E-06
1.5000E-01	1.9610E-05
5.0000E-02	2.3842E-06
-5.0000E-02	2.3246E-06
-1.5000E-01	1.9789E-05
-2.5000E-01	7.7486E-06
-3.5000E-01	6.1989E-06
-4.5000E-01	1.1802E-05
-5.5000E-01	1.7583E-05
-6.5000E-01	2.6226E-06
-7.5000E-01	2.0862E-06
-8.5000E-01	1.6212E-05
-9.5000E-01	7.0333E-06
-1.0500E+00	7.8678E-06
-1.1500E+00	1.1683E-05
-1.2500E+00	1.3709E-05
-1.3500E+00	3.6955E-06
-1.4500E+00	4.6492E-06
-1.5500E+00	1.8835E-05
-1.6500E+00	8.4639E-06
-1.7500E+00	1.9073E-06
-1.8500E+00	1.2338E-05
-1.9500E+00	1.7822E-05
-2.0500E+00	7.3910E-06
-2.1500E+00	3.9339E-06
-2.2500E+00	2.0266E-05
-2.3500E+00	3.6955E-06
-2.4500E+00	3.6359E-06

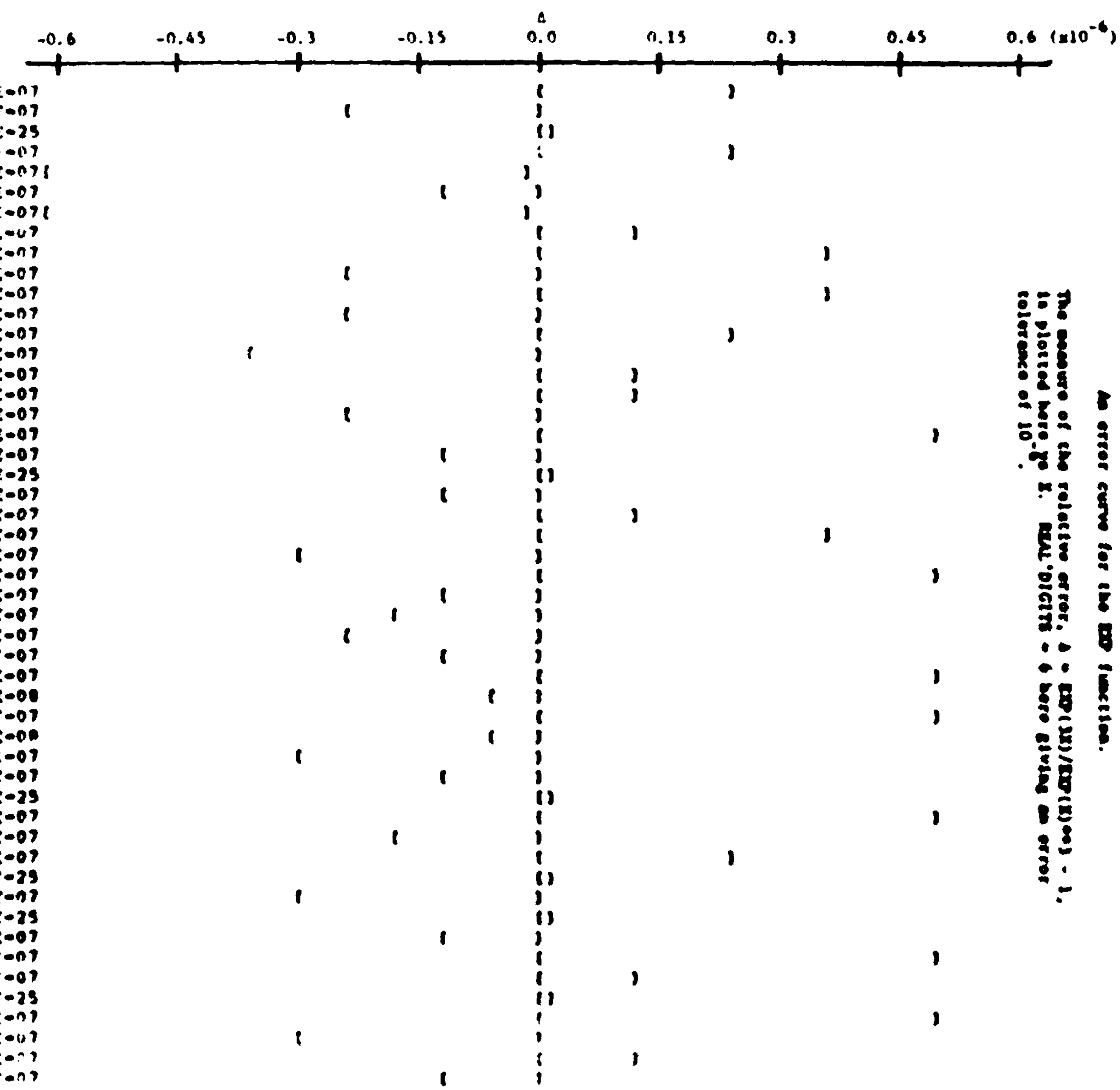


Execution complete
Execution time: 9605 seconds
I-code statements executed: 22985

Figure V.B-11

An error curve for the EXP function.

The measure of the relative error, Δ = EXP(3X)/EXP(X)**3 - 1, is plotted here vs X. REAL'DIGITS = 6 here giving an error tolerance of 10^{-6}.



3X	|Δ|
7.30000E+00	2.38419E-07
7.00000E+00	2.38419E-07
6.70000E+00	4.13590E-25
6.40000E+00	2.38419E-07
6.10000E+00	7.74860E-07
5.80000E+00	1.19209E-07
5.50000E+00	8.94070E-07
5.20000E+00	1.19209E-07
4.90000E+00	3.57628E-07
4.60000E+00	2.38419E-07
4.30000E+00	3.57628E-07
4.00000E+00	2.38419E-07
3.70000E+00	2.38419E-07
3.40000E+00	3.57628E-07
3.10000E+00	1.19209E-07
2.80000E+00	1.19209E-07
2.50000E+00	2.38419E-07
2.20000E+00	4.76837E-07
1.90000E+00	1.19209E-07
1.60000E+00	4.13590E-25
1.30000E+00	1.19209E-07
9.99998E-01	1.19209E-07
6.99998E-01	3.57628E-07
3.99998E-01	2.98023E-07
9.99984E-02	4.76837E-07
-2.00002E-01	1.19209E-07
-5.00002E-01	1.78814E-07
-8.00002E-01	2.38419E-07
-1.10000E+00	1.19209E-07
-1.40000E+00	4.76837E-07
-1.70000E+00	5.96046E-08
-2.00000E+00	4.76837E-07
-2.30000E+00	5.96046E-08
-2.60000E+00	2.98023E-07
-2.90000E+00	1.19209E-07
-3.20000E+00	4.13590E-25
-3.50000E+00	4.76837E-07
-3.80000E+00	1.78814E-07
-4.10000E+00	2.38419E-07
-4.40000E+00	4.13590E-25
-4.70000E+00	2.98023E-07
-5.00000E+00	4.13590E-25
-5.30000E+00	1.19209E-07
-5.60000E+00	4.76837E-07
-5.90000E+00	1.19209E-07
-6.20000E+00	4.13590E-25
-6.50000E+00	4.76837E-07
-6.80000E+00	2.98023E-07
-7.10000E+00	1.19209E-07
-7.40000E+00	1.19209E-07

Execution complete
Execution time: 10288 seconds
I-code statements executed: 24383

function; and of course similar tests must be run on the rest of the library functions before they can be used with any confidence.

V.C. Unresolved Problems

There is one major unsolved problem associated with the particular method of constructing function packages as given here. This has to do with memory space and the fact that Ada does not seem to have anything similar to the OVERLAY feature of FORTRAN. In setting up the elementary functions SIN, COS, EXP, etc., a set of Chebyshev constants are introduced for each function; these are used to calculate the coefficients of the power series or polynomial approximation. These new constants need to be retained for the remainder of the execution of the program, but the Chebyshev constants need not be retained; the space allocated to the Chebyshev coefficients could be abandoned or reused for something else, except that Ada has no mechanism for doing this. Also there are program segments that are needed only at initialization time; these too could be abandoned once initialization has been completed and the space reused, but again there is no mechanism for doing this.

In a paging environment on a large computer such as the VAX, this is no problem -- segments that are no longer needed are eventually rolled out and never brought back into main memory again. For the anticipated Ada applications however, say for executing an Ada program on a microprocessor with only 4K bytes, space is crucial and it is imperative that no longer needed data or program areas in memory be reusable. We do not know yet how in Ada to reuse either data or program memory area no longer needed, unless the system has some kind of automatic paging arrangement.

II. Results and Conclusions

The principal results of this effort show that the building of library components is well supported by the various Ada constructs such as packages, generics, procedures, functions and [illegible]. However, the present state of Ada compilers is [illegible] that certain aspects of Ada cannot yet be tested. In particular, one would like to show that the machine and accuracy independent techniques used here produce elementary math function routines that run as efficiently as hand written assembly language routines. Such comparisons cannot be made at this time and will depend extensively on the optimization effectiveness of the production quality Ada compiler. Assuming that the optimization procedures to be used are reasonably effective, the family of machine and accuracy independent routines described here should be as efficient as hand coded assembly language routines.

III Recommendations for Further Study

There are several tasks that need yet to be completed before a fully usable Ada library of elementary functions can be distributed for general use in a production environment.

1) The existing routines in the package presented here need to be rather thoroughly exercised and debugged. This was not done here because of the quality of the Ada compiler available and because of time limitations.

2) The packages here need to be compiled with a production quality Ada compiler and the resulting machine code be examined and compared to hand written assembly language routines for efficiency. This would be a measure both of the effectiveness of the Ada compiler and optimizer and of the elementary function routines themselves.

3) The routines written here are meant for floating point operations; similar routines [illegible] toward fixed point operations need to be written and examined in detail for efficiency.

4) The routines presented here are [illegible] for running time, [illegible] need to be examined. A compromise [illegible] lead to yet a different [illegible]

5) Finally, in an ideal Ada environment, the breadth of the function library needs to be considerably expanded and should also include, for example, functions of complex variables and values.

List of References

1. Dodes, Irving Allen, "Numerical Analysis for Computer Science." North-Holland, 1978. [illegible]

2. Hart, John F., et al., "Computer Approximations." John Wiley and Sons, 1968. [illegible]

3. Hastings, Cecil Jr., "Approximations for Digital Computers." Princeton University Press, 1955. [illegible]

4. Khovanskii, Alexey Nikolaevich, "The Application of Continued Fractions and Their Generalizations to Problems in Approximation Theory." P. Noordhoff, 1963. [illegible]

5. Ralston, Anthony, "A First Course in Numerical Analysis." McGraw-Hill, 1965. [illegible]

6. Rice, John R., "The Approximation of Functions." Addison-Wesley, 1964. [illegible]

7. Snyder, Martin Avery, "Chebyshev Methods in Numerical Approximation." Prentice-Hall, 1966. [illegible]

[illegible]

[illegible]
[illegible]

[illegible]
[illegible]
[illegible]

```
ADAfile:      Fun1.ADA
AISfile:      Fun1.AIS
LISfile:      Fun1.LIS

--      Disclaimer--    "This package of functions is an unvalidated
--      early version and is intended for experimental use only . . ."
--
--      Many of the constructs used in this package appear awkward and
--      unnatural but were necessary to overcome the flaws in the
--      Ada/ED Compiler Version 11.4, the only one available at the
--      time these packages were being prepared.
--
--                            TABLE I
--              A list of known flaws (as of 10/26/81) in
--              the Ada/ED Compiler Version 11.4
--
--      Compile time:
--          1.  Generic and actual parameters may not have the same name
--          2.  [illegible] not implemented
--          3.  [illegible] representation of a universal constant
--
--      Run time:
--          1.  [illegible] to a function [illegible] of value [illegible]
--          2.  [illegible] truncates instead of rounding (sometimes ?)
--          3.  Universal constants as parameters of functions [illegible]
--              [illegible] can cause a [illegible] time error halt, not a
--              [illegible] error.
--

generic type [illegible] is digits <>;
        type [illegible] is digits <>;


package [illegible] is
        type [illegible] is array(i [illegible] range <>) of [illegible];
        function [illegible] return REAL;
        function [illegible] return REAL;
        function [illegible]                return FLOAT;
        function [illegible]                return FLOAT;
        function [illegible]                return [illegible];
        function [illegible]                return [illegible];
        function [illegible]   return INTEGER;
        function [illegible]                return INTEGER;
        function [illegible]                return REAL;
        function [illegible]   return REAL;
        function [illegible]                return INTEGER;
        function [illegible]                return REAL;
        function [illegible] return [illegible];
        function [illegible] (X:REAL)  return REAL;
        function [illegible](A, [illegible])        return REAL;
        function [illegible], X:REAL)        return REAL;
        function [illegible]:REAL; [illegible]) return REAL;
        function [illegible](X:REAL; [illegible])      return REAL;
        function [illegible](X:REAL; [illegible])    return REAL;
```

```
            function EXP(X:REAL)      return REAL;
            function LN(X:REAL)       return REAL;
            function SQRT(X:REAL)     return REAL;
            function ATAN(X:REAL)     return REAL;
            function SIN(X:REAL)      return REAL;
            function COS(X:REAL)      return REAL;
            function TAN(X:REAL)      return REAL;
            function ASIN(X:REAL)     return REAL;
            function ACOS(X:REAL)     return REAL;
            function ATANH(X:REAL)    return REAL;
            function SINH(X:REAL)     return REAL;
            function COSH(X:REAL)     return REAL;
            function TANH(X:REAL)     return REAL;
            function ASINH(X:REAL)    return REAL;
            function ACOSH(X:REAL)    return REAL;

end [illegible];



package body [illegible] is


-- * is overloaded for I*R and
-- / is overloaded for R/I
-- where I is INTEGER and R is REAL or FLOAT or LONG_FLOAT.


function "*"(I:INTEGER; R:REAL) return REAL is --
-- pragma INLINE;
begin   return REAL(I)*R;
end "*";


function "/"(R:REAL; I:INTEGER) return REAL is --
-- pragma INLINE;
begin   return R/REAL(I);
end "/";


function "*"(I:INTEGER; R:FLOAT) return FLOAT is --
-- pragma INLINE;
begin   return FLOAT(I)*R;
end "*";


function "/"(R:FLOAT; I:INTEGER) return FLOAT is --
-- pragma INLINE;
begin   return R/FLOAT(I);
end "/";


function "*"(I:INTEGER; R:LONG_FLOAT) return LONG_FLOAT is --
-- pragma INLINE;
begin   return LONG_FLOAT(I)*R;
end "*";

```

```

function "/"(A:LONG_FLOAT; I:INTEGER) return LONG_FLOAT is --
-- pragma INLINE;
begin   return A/LONG_FLOAT(I);
end "/";


function BASE_2_EXPON(X:REAL) return INTEGER is --
-- pragma INLINE;
-- a function that gives the base 2 exponent
-- of X, such that 0.5 <= ABS(X)/2**BASE_2_EXPON(X) < 1.0,
-- (provided X is not 0);
-- it should be machine dependent and written in assembly code.
-- pragma INLINE;
        Z:REAL := ABS(X);
        J:INTEGER := 0;
begin   if X /= 0.0 then
                while Z >= 1.0
                loop    Z := Z/2;
                        J := J + 1;
                end loop;
                while Z < 0.5
                loop    Z := 2*Z;
                        J := J - 1;
                end loop;
        end if;
        return J;
end BASE_2_EXPON;


TWO:constant INTEGER := 2; -- needed because of bug in NYU compiler versi

function TWO_POWER_INT(J:INTEGER) return INTEGER is --
-- pragma INLINE;
-- a function equivalent to 2**J;
-- it would be quickest in assembly code.
begin   return TWO**J;
end TWO_POWER_INT;


function TWO_POWER_REAL(I:INTEGER) return REAL is --
-- pragma INLINE;
-- a function equivalent to REAL(TWO**I);
-- it would be quickest in assembly code.
begin   return REAL(TWO**I);
end TWO_POWER_REAL;


function COMPOSE_REAL(X:REAL) return REAL is --
-- pragma INLINE;
-- a function equivalent to X/TWO_POWER_REAL(BASE_2_EXPON(X));
-- it would be quickest in assembly code.
begin   return X/REAL(TWO**BASE_2_EXPON(X));
end COMPOSE_REAL;


-----------------------------------------------------------------------
```

```
--
-- UTILITIES:
-- some of these could best be written is assembly code.
--
-----------------------------------------------------------------------


function COND(
        B:BOOLEAN;
        X,Y:REAL)
        return REAL is --
-- pragma INLINE;
begin   if B then return X;
        else return Y;
        end if;
end COND;


function COND(
        B:BOOLEAN;
        X,Y:INTEGER)
        return INTEGER is --
-- pragma INLINE;
begin   if B then return X;
        else return Y;
        end if;
end COND;


function ODD(I:INTEGER) return BOOLEAN is --
-- pragma INLINE;
-- Better in assembly code.
begin   return I mod 2 /= 0;
end ODD;


function ROUND(X:REAL) return INTEGER is --
-- pragma INLINE;
-- NOTE: Version 11.4 of compiler TRUNCATES TOWARDS ZERO
--       instead of rounding for INTEGER.
begin   if X > 0.0 then return INTEGER(X + 0.5);
        else    return INTEGER(X - 0.5);
        end if;
end ROUND;


function "rem"(A, B:REAL) return REAL is --
-- pragma INLINE;
-- Better in assembly if there is a machine
-- code instruction for remainder divide of FLOATs.
begin   return(A - B*REAL(ROUND(A/B)));
end "rem";


function SIGN(Z, X:REAL) return REAL is --
-- pragma INLINE;
begin   if Z < 0.0 then return -X;
```

```
            elsif Z > 0.0 then return X;
            else    return 0.0;
            end if;
    end SIGN;


    function POLY(X:REAL; A:A_REAL) return REAL is --
    -- pragma INLINE; -- ?
            TEMP:REAL := A(A'LAST);
    begin   for J in reverse 0..A'LAST-1
            loop    TEMP := A(J) + X*TEMP;
            end loop;
            return TEMP;
    end POLY;


    function ODD_POLY(X:REAL; A:A_REAL) return REAL is --
    -- pragma INLINE;
    begin   return X*POLY(X*X, A);
    end ODD_POLY;


    function EVEN_POLY(X:REAL; A:A_REAL) return REAL is --
    -- pragma INLINE;
    begin   return POLY(X*X, A);
    end EVEN_POLY;



    ----------------------------------------------------------------------
    --
    --              The Elementary Functions in ADA
    --
    --      In the past the elementary functions have been written in
    --  assembly code and incorporated into the higher order language
    --  program by being intrinsics of the language and were handled as
    --  part of the compiler.  The accuracy was fixed at the accuracy of
    --  the machine for which the compilation was done.  In Ada, however,
    --  neither the machine nor the accuracy is known until compile time.
    --      In most cases these functions are mapped onto some reduced range
    --  and then approximated over that range by a Chebyshev polynomial
    --  derived power series.  (The advantage of the Chebyshev polynomials
    --  are that they give a smaller maximum error over the approximation
    --  range than any other polynomial of the same degree.)
    --      The Chebyshev polynomials themselves are not used but the sums
    --  of their coefficients of like powers of the variable are used as
    --  the constants in the power series.  Since the Chebyshev polynomials
    --  contain all powers up to the order of the polynomial, adding one
    --  more term (to get more accuracy for example) changes all the terms
    --  in the power seres.  This means that the number and values of the
    --  coefficients must be calculated at compile time (or later) when
    --  required accuracy is known.
    --
    --
    ----------------------------------------------------------------------


```

```

subtype INT_1_2 is INTEGER range 1..2;
subtype INT_0_1 is INTEGER range 0..1;
KMAX:constant := 10;
type A_LONG_FLOAT is array(INTEGER range <>) of LONG_FLOAT;


PI       :constant := 3.14159_26535_89793_23846_26433;
SQRT_2   :constant := 1.41421_35623_73095_04880_16887;
LN_2     :constant := 0.69314_71805_59945_30941_72321;


-----------------------------------------------------------------
--
--        CHEBYSHEV EXPANSION
--
--
-- Let
--        f(x) =  Sum(B(k)*T(k,x/d)),      -d<=x<=d
--              (0<=k<=n)
--
-- where
--        T(k,x/d) =      Sum(C(k,j)*(x/d)**j)
--                       (0<=j<=k)
--
--        B(k) =  (2/pi)*Integral(f(x)*T(k,x/d)/sqrt(1-(x/d)**2)),  k>0
--                       (-1<=x/d<=1)
--
--              =  (2/pi)*Integral(f(d*cos(y))*cos(k*y))
--                       (0<=y<=pi)
--
-- then
--        f(x) =  Sum(       Sum(B(k)*C(k,j)*(x/d)**j))
--              (0<=k<=n) (0<=j<=k)
--
--              =  Sum(A(j)*x**j)
--              (0<=j<=n)
--
-- where
--        A(j) =  (1/d)**j*Sum(B(k)*C(k,j))
--                         (0<=k<=n)
--
--
-----------------------------------------------------------------

package REAL_IO is new FLOAT_IO(REAL);
use REAL_IO;


procedure ICH_SUM(
        A:out A_REAL;
        B:A_LONG_FLOAT;
        L:INT_1_2;
        I:INT_0_1) is --
        --  L = 2,  I = 0  gives even power series,
        --  L = 2,  I = 1  gives odd power series,
        --  L = 1,  I = 0  gives both even and odd terms,
```

```
          --  L = 1,  I = 1  is undefined.
          -- The CHEBYSHEV constants:
          function C(N, J:INTEGER) return INTEGER is --
                  --  This is not efficient, time wise, but no matter;
                  --  only used to initialize package constants,
                  --  never used in real time.
                  --  This IS efficient space wise, since it eliminates
                  --  the need of storing the sparse matrix C(N, J).
          begin   if N < 0 or else J < 0 or else J > N
                  or else ODD(N+J) then return 0;
                  elsif (J=0 and N=0) or else (N=1 and J=1) then return 1;
                  else return 2*C(N-1,J-1) - C(N-2, J);
                  end if;
          end C;
          TEMP:LONG_FLOAT;
          CTEMP:INTEGER;
  begin   -- TCH_SUM
          for J in 0..A'LAST
          loop    TEMP := 0.0;
                  for K in J..A'LAST
                  loop    CTEMP := C(L*K+I, L*J+I);
                          --  Version 11.4 wont multyply by floating zero.
                          if CTEMP /= 0 then
                                  TEMP := TEMP + CTEMP*B(K);
                          end if;
                  end loop;
                  A(J) := REAL(TEMP*B(-1)**(-L*J-I));
                  PUT(A(J), WIDTH => 15);
          end loop;
          new_line;
  end TCH_SUM;


  function LIM(B:A_LONG_FLOAT; G:REAL := 10.0) return INTEGER is --
          K:INTEGER := 1;
          REAL_DIGITS:INTEGER := INTEGER(REAL'DIGITS);
  begin
          PUT(G, WIDTH => 15);  new_line;
          PUT(2*G, WIDTH => 15);  new_line;
          loop    if ABS(REAL(B(K))) < G**(-REAL_DIGITS) then
                          return K - 1;
                  elsif K = B'LAST then return K;
                  end if;
                  K := K + 1;
          end loop;
  end LIM;


  -- Note B_xxx(-1) = magnitude of range of variable in function xxx;
  -- ie
  --       B_SIN(-1)       = pi/4           B_COS(-1)        = pi/4
  --       B_ASIN(-1)      = 0.375          B_TAN(-1)        = pi/8
  --       B_SQRT(-1)      = 0.25           B_EXP(-1)        = 0.25
  --       B_ATAN(-1)      = sqrt(2) - 1
  --       B_LN(-1)        = (2 - sqrt(2))/(2 + sqrt(2))
  --       B_SINH(-1)      = 1.0            B_COSH(-1)       = 1.0
  --       B_TANH(-1)      = 0.5            B_ATANH(-1)      = 0.25
```

```
--       B_ASINH(-1)      = 0.375


-- These constants are believed to be accurate to the 24th decimal digit.

B_SIN:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
PI/4,                                  0.94977_04415_68744_77636_82691,
-0.04984_04113_37036_66401_49298,      0.00038_77134_36152_82730_90288,
-0.00000_14305_80091_93208_96335,      0.00000_00030_73651_15544_85672,
-0.00000_00000_04316_36597_42291,      0.00000_00000_00004_27564_99507,
-0.00000_00000_00000_00314_36072,      0.00000_00000_00000_00000_17840,
-0.00000_00000_00000_00000_00008,      0.0);

B_COS:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
PI/4,                                  0.85163_19137_04808_01270_04060,
-0.14643_06443_90336_86332_07964,      0.00192_14493_11814_64679_69079,
-0.00000_99649_08489_82930_00687,      0.00000_00275_76595_60718_73952,
-0.00000_00000_47399_49806_16484,      0.00000_00000_00055_49548_54157,
-0.00000_00000_00000_04709_70491,      0.00000_00000_00000_00003_02990,
-0.00000_00000_00000_00000_00153,      0.0);

B_TAN:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
PI/8,                                  1.02695_66055_75433_75318_77491,
 0.02738_60540_75084_38492_31238,      0.00043_63685_44497_59071_23490,
 0.00000_70316_73577_10477_88349,      0.00000_01134_29567_27361_97971,
 0.00000_00018_23961_53556_18653,      0.00000_00000_29523_14912_17757,
 0.00000_00000_00475_30369_23712,      0.00000_00000_00007_68431_70544,
 0.00000_00000_00000_12337_28568,      0.00000_00000_00000_00200_00828);

B_ATAN:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
 0.41421_35623_73095_04880_16887,      0.97341_02303_44744_14048_51820,
-0.02595_27204_82638_35037_97975,      0.00361_89529_74923_69926_44632,
-0.00031_75374_93335_65973_14001,      0.00000_05405_76654_50161_32801,
-0.00000_00175_13588_81541_58313,      0.00000_00005_87028_14861_02751,
-0.00000_00000_23112_43147_91343,      0.00000_00000_00703_55322_00059,
-0.00000_00000_00024_91546_52757,      0.00000_00000_00000_89227_36557);

B_ASIN:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
 0.375,                                1.01731_73145_25931_81726_33231,
 0.01252_12424_80649_02473_23989,      0.00021_15961_88859_04819_81129,
 0.00000_47408_83198_71353_39921,      0.00000_01219_88403_35586_17913,
 0.00000_00033_94473_34940_65333,      0.00000_00000_99537_06346_24424,
 0.00000_00000_03029_54601_23204,      0.00000_00000_00094_82716_48857,
 0.00000_00000_00013_03226_28731,      0.00000_00000_00000_09563_79223);

B_SINH:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
 1.0,                                  1.01852_10970_23599_61513_79419,
 0.03759_32219_22751_47715_49002,      0.00107_94777_73567_13275_92427,
 0.00000_63748_49260_75475_04916,      0.00000_00223_23863_04923_05362,
 0.00000_00009_49879_40140_41585,      0.00000_00000_00000_30571_73053_55412,
 0.00000_00000_00000_00473_15871,      0.00000_00000_00000_00006_69349,
 0.00000_00000_00000_00000_00635,      0.0);

B_COSH:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
 1.0,
 1.26606_58777_52008_33559_82446,      0.27149_53395_37876_56236_57050,
 0.00547_42404_32043_13265_02744,      0.00000_49773_27053_29514_00837,
```

```
     0.00000_01092_12169_60727_95726,    0.00000_00005_50589_60796_73746,
     0.00000_00006_01037_15223_06775,    0.00000_00000_00001_42375_80163,
     0.00000_00000_00000_00138_01801,    0.00000_00000_00000_00000_12074,
     0.00000_00000_00000_00000_00038);

    B_TAN:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
     0.5,                                0.96121_62263_72739_36911_57321,
    -0.03766_11413_53411_75513_26219,    0.00090_04074_71148_92802_75877,
    -0.00102_16447_00571_96941_59411,    0.00000_05231_11543_14572_47830,
    -0.00030_00126_14754_17044_56745,    0.00000_00003_04401_07642_67180,
    -0.00000_00000_07343_05191_12964,    0.00000_00000_00177_13606_90129,
    -0.00000_00000_00004_27331_50638,    0.00000_00000_00000_10307_84646);

    B_ATAN:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
     0.25,                               1.01072_10205_68314_61394_26297,
     0.01082_44195_16700_60924_64941,    0.00010_45880_96559_30385_36636,
     0.00000_12040_22043_43634_59001,    0.00000_00150_98397_67960_61666,
     0.00000_00001_97206_56850_03680,    0.00000_00000_02718_54483_61932,
     0.00000_00000_00038_00154_51119,    0.00000_00000_00000_54085_29074,
     0.00000_00000_00000_00780_59602,    0.00000_00000_00000_00011_39265);

    B_ASIN:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
     0.375,                              0.94880_16100_57773_90873_57081,
    -0.01103_06268_61208_03044_08235,    0.00016_44563_41614_66395_75926,
    -0.00000_32323_74655_32207_31002,    0.00000_00725_29159_60438_06842,
    -0.00000_00017_59429_72940_49692,    0.00000_00000_44935_08066_33908,
    -0.00000_00000_01190_32907_91532,    0.00000_00000_00032_40408_37609,
    -0.00000_00000_00106_90099_55232,    0.00000_00000_00000_02547_89448);

    B_EXP:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
     0.25,                               1.19815_13547_68553_88642_78879,
     0.20084_83645_34926_29734_13946,    0.00894_98316_92862_14978_54488,
     0.00025_83197_33596_70092_23755,    0.00000_55933_25892_90406_60652,
     0.00000_00969_00705_10794_31033,    0.00000_00013_99051_06341_84259,
     0.00000_00000_17314_61306_18070,    0.00000_00000_00187_50506_43298,
     0.00000_00000_00001_80436_89915,    0.00000_00000_00000_01563_77974);

    B_LN:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
     0.17157_28752_53809_90239_66225,    1.00497_23620_80994_03273_56872,
     0.00499_46092_97031_09833_96480,    0.00002_23658_03341_91784_63274,
     0.00000_01192_75852_10274_22078,    0.00000_00006_92753_58723_18746,
     0.00000_00000_04232_91471_88321,    0.00000_00000_00026_75000_48798,
     0.00000_00000_00000_17315_16241,    0.00000_00000_00000_00114_11170,
     0.00000_00000_00000_00000_76281,    0.00000_00000_00000_00000_00515);

    B_SQRT:constant A_LONG_FLOAT(-1..KMAX) := (  -- Initialize:
     0.25,                               0.85984_66001_02237_79135_66373,
     0.14590_57348_49680_61465_28198,   -0.00623_51235_98338_35862_41206,
     0.00053_38902_15777_14176_08675,   -0.00005_71863_30329_11416_87631,
     0.00000_68629_79719_25193_60268,   -0.00000_08826_51603_94746_00220,
     0.00000_01189_39867_39664_64765,   -0.00000_00165_75365_75860_25774,
     0.00000_00023_69307_19312_85914,   -0.00000_00003_45461_45137_37718);


    B_SIN_RAN        : constant REAL := REAL(B_SIN(-1));
    B_COS_RAN        : constant REAL := REAL(B_COS(-1));
    B_TAN_RAN        : constant REAL := REAL(B_TAN(-1));
```

```
R_ATAN_RAN        : constant REAL := REAL(R_ATAN(-1));
R_ASIN_RAN        : constant REAL := REAL(R_ASIN(-1));
R_EXP_RAN         : constant REAL := REAL(R_EXP(-1));
R_LN_RAN          : constant REAL := REAL(R_LN(-1));
R_SQRT_RAN        : constant REAL := REAL(R_SQRT(-1));
R_SINH_RAN        : constant REAL := REAL(R_SINH(-1));
R_COSH_RAN        : constant REAL := REAL(R_COSH(-1));
R_TANH_RAN        : constant REAL := REAL(R_TANH(-1));
R_ATANH_RAN       : constant REAL := REAL(R_ATANH(-1));
R_ASINH_RAN       : constant REAL := REAL(R_ASINH(-1));


P_SIN   : A_REAL(0..LIM(R_SIN));
P_COS   : A_REAL(0..LIM(R_COS));
P_TAN   : A_REAL(0..LIM(R_TAN));
P_ATAN  : A_REAL(0..LIM(R_ATAN));
P_ASIN  : A_REAL(0..LIM(R_ASIN));
P_EXP   : A_REAL(0..LIM(R_EXP));
P_LN    : A_REAL(0..LIM(R_LN));
P_SQRT  : A_REAL(0..LIM(R_SQRT, 3.16));          --  3.16 = sq (10)
P_SINH  : A_REAL(0..LIM(R_SINH));
P_COSH  : A_REAL(0..LIM(R_COSH));
P_TANH  : A_REAL(0..LIM(R_TANH));
P_ATANH : A_REAL(0..LIM(R_ATANH));
P_ASINH : A_REAL(0..LIM(R_ASINH));


-----------------------------------------------------------------
--
--         Trigonometric Transformations
--
--         sin x    =  sin(x mod 2pi)
--                  =  sign(x) sin(pi - |x|)
--                  =  sign(x) cos(pi/2 - |x|)
--                  =  sign(x) (cos(|x| - pi/4) + sin(|x| - pi/4))/sqrt(2)
--
--         cos x    =  cos(x mod 2pi)
--                  =  - cos(pi - |x|)
--                  =  sin(pi/2 - |x|)
--                  =  (cos(|x| - pi/4) - sin(|x| - pi/4))/sqrt(2)
--
--         tan x    =  tan(x mod 2pi)
--                  =  sign(x) tan(|x| - pi)
--                  =  sign(x)/tan(pi/2 - |x|)
--                  =  sign(x)(tan(|x| - pi/4) + 1)/(1 - tan(|x| - pi/4))
--                  =  sign(x)(tan(|x|-b) + tan(b))/(1 - tan(b)tan(|x|-b))
--                  =  2tan(x/2)/(1 - tan(x/2)**2)
--
--         asin x   =  atan(x/sqrt(1 - x**2))
--
--         acos x   =  pi/2 - asin(x)
--                  =  sqrt(2y + y**2/3 + 4y**3/45 +   .   .   .   )
--                     where y = 1 - x
--
--
--         Chebyshev Series Used
--
```

```
--      ln x    =  -ln(2)/2 + 2((x - r)/(x + r) +
--                  (1/3)((x - r)/(x + r))**3 +
--                  (1/5)((x - r)/(x + r))**5 +  .  .  .  )
--      where r = 1/sqrt(2).
--
--
--      The hyperbolics
--
--      sinh x  =  (exp(x) - 1/exp(x))/2
--
--      cosh x  =  (exp(x) + 1/exp(x))/2
--
--      tanh x  =  sinh(x)/cosh(x)
--
--      exp x   =  sinh(x) + cosh(x)
--
--      atanh x =  ln((1 + x)/(1 - x))/2
--
--      asinh      [illegible](ln(x) + sqrt(1 + x**2))
--
--      [illegible] x =  [illegible] + [illegible]
--                =  [illegible] + [illegible]/45 +  .  .  .  )
--                  [illegible]
--
-----------------------------------------------------------------


--      In addition, execution times for a function the best, average,
--      and worst case are given.  The average is based on an assumed
--      uniform distribution of either the variable (for exp, ln, sqrt,
--      or atan) or the value of the function (inverse trig functions)
--      over an appropriate interval.
--      The A, M, D variables in these expressions represent the
--      (limits of) arithmetic times for add, multiply, and divide.
--      The times for other operations are assumed negligible.
--      The number of squares in the formula stands for the number of
--      terms needed in the polynomial approximation (varies with the
--      accuracy required).  If the function calls other functions
--      then these costs are excluding those other functions which
--      will be so designated (ie sqrt, etc . . . ).  Where available,
--      tables from Hart, et al. are given of n vs log of the error.
--      These error tables are consistent with our own
--      calculations.


[illegible]
SQRT_[illegible]


-----------------------------------------------------------------
--
--      [illegible]
--
-----------------------------------------------------------------

function [illegible] return REAL is --
-- [illegible] --
```

```
          [illegible] is [illegible];
          [illegible] is [illegible];
          [illegible] is [illegible];
begin     -- assert [illegible]
          [illegible] := [illegible];
          if [illegible] then [illegible];
          end if;
          if [illegible] then [illegible];
          end if;
          if [illegible] then [illegible];
          end if;
          return [illegible];
          [illegible]
                    return [illegible];
end [illegible];
--        [illegible]
--        [illegible]
--        [illegible]
--        [illegible]

--        [illegible]
--        [illegible]
--   [illegible]


-------------------------------------------------------------------------
--
--        [illegible]
--
-------------------------------------------------------------------------

function [illegible] return REAL is --
-- [illegible] --
          [illegible] is [illegible];
          [illegible] is [illegible];
begin     -- assert [illegible]
          if [illegible] then return [illegible];
          end if;
          -- assert [illegible]
          return [illegible]
                    [illegible];
end [illegible];
--        [illegible]
--        [illegible]

--        [illegible]
--        [illegible]
--   [illegible]


-------------------------------------------------------------------------
--
--        [illegible]
--
-------------------------------------------------------------------------

function [illegible] return REAL is --
```

```
-- ... appropriate when it
-- ... large.
...
... ABS(Z), F_TINY));
...
-- ... appropriate when it
-- ... is very large.
...
-----------------------------------------------------------------------
--
--
```

```
--
------------------------------------------------------------------------

function SIN(X:REAL) return REAL is --
-- pragma INLINE; -- ?
        Z:REAL := X;
begin   loop --
                if ABS(Z) <= 3*PI/4 then return SIN_SMALL(Z);
                elsif ABS(Z) <= PI then Z := SIGN(Z, PI - ABS(Z));
                else Z := Z rem (2*PI);
                end if;
        end loop;
end SIN;
--      Computation time:
--      Best:   2M + (A + M)n
--      Ave:    D + 0.5M + (A + M)(n + 1)
--      Worst:  D + M + (A + M)(n + 1)

--      Hart, page 117 (range[0, pi/4]):
--              n =     3     4     5     6     7
--   -log10(err) =   8.5  11.3  14.3  17.5  20.7


------------------------------------------------------------------------
--
--      COS
--
------------------------------------------------------------------------

function COS(X:REAL) return REAL is --
-- pragma INLINE;
begin   if ABS(X) <= 3*PI/4 then return COS_SMALL(X);
        end if;
        return SIN(PI/2 - ABS(X));
end COS;
--      Computation time:
--      Best:   M + (A + M)n
--      Ave:    D + A + 0.5M + (A + M)(n + 1)
--      Worst:  D + (A + M)(n + 2)

--      Hart, page 118 (range[0, pi/2]):
--              n =     3     4     5     6     7
--   -log10(err) =   7.5  10.3  13.2  16.3  19.5


------------------------------------------------------------------------
--
--      TAN
--
------------------------------------------------------------------------

function TAN(X:REAL) return REAL is --
-- pragma INLINE; -- ?
        Z:REAL := X;
begin   loop --
                if ABS(Z) <= 5*PI/8 then return TAN_SMALL(Z);
                elsif ABS(Z) <= PI then Z := SIGN(Z, ABS(Z) - PI);
```

```
                        [illegible]   := [illegible];
                        [illegible];
                [illegible];
    [illegible];
--              [illegible]:
--          Best:    2 + (A + 5)n
--          Ave:     1.[illegible] - [illegible].25A + ([illegible] + [illegible])(n + 2)
--          Worst:   [illegible] + [illegible] + (A + 5)(n + 2)

--          [illegible] (range[0, [illegible]]):
--                      n =  1     2     3     4     5     6     7     8
--     -log10(err) = [illegible]   5.2   [illegible]   [illegible]   1[illegible].5  12.3  14.1  15.9


----------------------------------------------------------------------
--
--          ATAN
--
----------------------------------------------------------------------

function ATAN(X:REAL) return REAL is --
-- pragma INLINE; -- ?
            Z:REAL := ABS(X);
begin       if Z <= [illegible]_ATAN_RAN then
                    return ODD_POLY(X, P_ATAN);
            elsif Z <= 1.0/[illegible]_ATAN_RAN then
                    return SIGN(X,PI/4+ODD_POLY((Z-1.0)/(Z+1.0),P_ATAN));
            end if;
            return SIGN(X, PI/2 - ODD_POLY(1.0/Z, P_ATAN));
end ATAN;
--          Computation time:
--          Best:    2* + (A + M)n
--          Ave:     [illegible].75(A + [illegible]) + [illegible] + (A + M)(n + 1)
--          Worst:   [illegible] + A + (A + M)(n + 2)

--          Hart, page 129 (range[0, tan PI/8])*:
--                  n =  2     3     4     5     6     7     8     9     10    11
--   -log10(err) = 4.7   6.3   7.7   9.2   10.7  12.1  13.6  15.0  16.5  17.9

--   * the range [0, sqrt(2) - 1]


----------------------------------------------------------------------
--
--          ASIN
--
----------------------------------------------------------------------

function ASIN(Y:REAL) return REAL is --
-- pragma INLINE; -- ?
begin       -- assert ABS(Y) <= 1.0;
            if ABS(Y) < R_ASIN_RAN then return ODD_POLY(Y, P_ASIN);
            elsif ABS(Y) >= 1.0 then return SIGN(Y, PI/2);
            end if;
            return ATAN(Y/SQRT((1.0 + Y)*(1.0 - Y)));
end ASIN;
--          Computation time:
```

```
--        Best:    2D + (A + M)n
--        Ave:     2.0625D + 0.5625A + 1.25M +
--                 0.75(A + M)(n[atan] + n[sqrt] + n/3 + 3)
--        Worst:   3D + A + (A + M)(n[atan] + n[sqrt] + 5)


----------------------------------------------------------------------
--
--        ACOS
--
----------------------------------------------------------------------

ACOS_CONST:REAL;

function ACOS(U:REAL) return REAL is --
-- pragma INLINE;
          Y:REAL := 1.0 - U;
begin     -- assert ABS(U) <= 1.0;
          if ABS(Y) < ACOS_CONST then return SQRT(Y*(2.0 + Y/3));
          end if;
          return PI/2 - ASIN(U);
end ACOS;
--        Computation time:
--        Best:    D + 2A + (A + M)(n[sqrt] + 3)
--        Ave:     2.0625D + 2.5625A + 1.25M +
--                 0.75(A + M)(n[atan] + n[sqrt] + n[asin]/3 + 4)
--        Worst:   3D + 3A + (A + M)(n[atan] + n[sqrt] + 5)


----------------------------------------------------------------------
--
--        SINH
--
----------------------------------------------------------------------

function SINH(X:REAL) return REAL is --
-- pragma INLINE; -- ?
          E:REAL;
begin     if ABS(X) <= P_SINH_RAN then return ODD_POLY(X, P_SINH);
          end if;
          E := EXP(X);
          if E = 0.0 then return -REAL(REAL'LARGE)/2;
          end if;
          return (E - 1.0/E)/2;
end SINH;

--        Hart, page 104 (range[0,0.5]):
--                    n =  0     1     2     3     4     5     6     7
--   -log10(err) = 1.7   4.2   7.0   10.1  13.3  16.7  20.3  23.9
--        Range[0, 1.0]:
--                    =          3.0   5.2   7.7   10.3  13.1  16.0  19.1


----------------------------------------------------------------------
--
--        COSH
--
```

```
---------------------------------------------------------------------

function COSH(X:REAL) return REAL is --
-- pragma INLINE; -- ?
        E:REAL;
begin   if ABS(X) <= R_COSH_RAN then return EVEN_POLY(X, P_COSH);
        end if;
        E := EXP(X);
        if E = 0.0 then return REAL(REAL'LARGE)/2;
        end if;
        return (E + 1.0/E)/2;
end COSH;

--      Range[0, 1.0]:
--            n =  1    2    3    4     5     6     7
--   -log10(err) = 2.3  4.3  6.7  9.3  12.0  14.8  17.8


---------------------------------------------------------------------
--
--      TANH
--
---------------------------------------------------------------------

TANH_CONST:REAL;

function TANH(X:REAL) return REAL is --
-- pragma INLINE; -- ?
        E2X:REAL;
begin   if ABS(X) <= R_TANH_RAN then return ODD_POLY(X, P_TANH);
        elsif ABS(X) >= TANH_CONST then return SIGN(X, 1.0);
        end if;
        E2X := EXP(2*X);
        return (E2X - 1.0)/(E2X + 1.0);
end TANH;


---------------------------------------------------------------------
--
--      ATANH
--
---------------------------------------------------------------------

function ATANH(X:REAL) return REAL is --
-- pragma INLINE; -- ?
begin   -- assert ABS(X) < 1.0;
        if ABS(X) <= R_ATANH_RAN then return ODD_POLY(X, P_ATANH);
        elsif ABS(X) >= 1.0  then return SIGN(X, LN_REAL_LARGE/2);
        end if;
        return LN((1.0 + X)/(1.0 - X))/2;
end ATANH;

--      Range[0, 0.25]:
--            n =  1    2    3    4     5     6     7     8
--   -log10(err) = 3.0  5.1  7.2  9.7  11.6  13.4  15.3  17.1
```

```
----------------------------------------------------------------------
--
--      ASINH
--
----------------------------------------------------------------------

function ASINH(X:REAL) return REAL is --
-- pragma INLINE; -- ?
begin   if ABS(X) <= P_ASINH_RAN then return ODD_POLY(X, P_ASINH);
        elsif ABS(X) >= SQRT_REAL_LARGE/2 then
                return SIGN(X, LN(ABS(X)) + LN_2);
        end if;
        return SIGN(X, LN(ABS(X) + SQRT(1.0 + X*X)));
end ASINH;

--      Range[0, 0.375]:
--              n =  2    3    4     5     6     7     8     9    10
--  -log10(err) = 5.5  7.1  8.8  10.3  11.1  13.5  15.0  16.6  18.1


----------------------------------------------------------------------
--
--      ACOSH
--
----------------------------------------------------------------------

ACOSH_CONST:REAL;

function ACOSH(U:REAL) return REAL is --
-- pragma INLINE; -- ?
        Y:REAL := U - 1.0;
begin   -- assert U >= 1.0;
        if U <= 1.0 then return 0.0;
        elsif Y <= ACOSH_CONST then return SQRT(Y*(2.0 - Y/3));
        elsif U >= SQRT_REAL_LARGE/2 then return LN(ABS(U)) + LN_2;
        end if;
        return LN(SQRT(Y*(1.0 + U)) + U);
end ACOSH;


TWO_P_FIVE: constant REAL := 2.5; -- Needed by compiler version 11.4.

begin   --  Initialization of package ELE_FUN.


        TCH_SUM(P_SIN,  B_SIN,  2, 0);
        TCH_SUM(P_COS,  B_COS,  2, 0);
        TCH_SUM(P_TAN,  B_TAN,  2, 0);
        TCH_SUM(P_ASIN, B_ASIN, 2, 0);
        TCH_SUM(P_ATAN, B_ATAN, 2, 0);
        TCH_SUM(P_EXP,  B_EXP,  1, 0);
        TCH_SUM(P_SINH, B_SINH, 2, 0);
        TCH_SUM(P_COSH, B_COSH, 2, 0);
        TCH_SUM(P_TANH, B_TANH, 2, 0);
        TCH_SUM(P_ASINH,B_ASINH,2, 0);
        TCH_SUM(P_ATANH,B_ATANH,2, 0);
        TCH_SUM(P_LN,   B_LN,   2, 0);
```

```
            TCH_SUM(P_SQRT, Q_SQRT, 1, 0);


            for I in 0..LIM(B_LN)
            loop    P_LN(I) := 2*P_LN(I);
            end loop;

            ACOS_CONST := 4.0/TWO_P_FIVE**REAL'DIGITS;
            ACOSH_CONST := 4.0/TWO_P_FIVE**REAL'DIGITS;
            TANH_CONST := REAL(REAL'DIGITS)*1.2 + LN_2;  -- 1.2 ~= ln(10)/2

            PUT(ACOS_CONST, WIDTH => 15);
            PUT(ACOSH_CONST, WIDTH => 15);
            PUT(TANH_CONST, WIDTH => 15);
            new_line;

            LN_REAL_LARGE := LN(REAL(REAL'LARGE));
            SQRT_REAL_LARGE := SQRT(REAL(REAL'LARGE));

            PUT(LN_REAL_LARGE, WIDTH => 15);
            PUT(SQRT_REAL_LARGE, WIDTH => 15);
            new_line;

    end ELE_FUN;


    ------------------------------------------------------------------------
    --
    --      REFERENCES
    --
    --      Rice, John R., "The Approximation of Functions",
    --      Addison-Wesley, 1964.  QA221 .R5 v1
    --
    --      Hastings, Cecil Jr, "Approximations for Digital Computers",
    --      Princeton University Press, 1955.  QA76 .H37 c.2
    --
    --      Ralston, Anthony, "A First Course  in Numerical Analysis",
    --      McGraw-Hill, 1965.  QA297 .R3
    --
    --      Snyder, Avery Martin, "Chebyshev Methods in Numerical
    --      Approximation", Prentice-Hall, 1966.  QA221 .S65
    --
    --      Khovanskii, Alexey Nikolaevitch, "The Application of Continued
    --      Fractions and Their Generalizations to Problems in
    --      Approximation Theory", P. Noordhoff, 1963.  QA221 .K473
    --
    --      Hart, John F., et al., "Computer Approximations," John
    --      Wiley and Sons, 1968.  QA297 .C64
    --
    --      Dodes, Irving Allen, "Numerical Analysis for Computer
    --      Science," North-Holland, 1978.  QA297 .D59
    --
    ------------------------------------------------------------------------

    with ELE_FUN;

    procedure TEST is
```

```
            type REEL is digits 5;
            type LONG_FLOOT is digits 7;
            package NEW_ELE_FUN is new ELE_FUN(REEL, LONG_FLOOT);
            use NEW_ELE_FUN;
  begin     -- TEST
            PUT(" TEST OK ");
            new_line;
  end TEST;
```

no parse errors detected
Parsing time: 732 seconds

no semantic errors detected
Translation time: 2236 seconds

APPENDIX B

A Listing of the Programs and Subroutines for Computing the Chebyshev Coefficients for the Various Elementary Math Functions

These programs are in FORTRAN but for documentation purposes, the FLECS listings are first given. The numbers in the right hand column of the FORTRAN listings show the line to which it corresponds in the FLECS listing.

COPY,DSUBOE

```
C
C      THE PROGRAM TRIGOE CALCULATES THE CHEBYSHEV AND POLYNOMIAL
C      COEFFICIENTS FOR ODD OR EVEN (MOSTLY TRIG) FUNCTIONS.
C      ALL COMPUTATIONS ARE IN DOUBLE PRECISION EXCEPT FOR THE
C      ERROR TOLERANCE WHICH IS IN SINGLE.
C      THE ACTUAL COMPUTATIONS ARE CARRIED OUT IN SUBROUTINE POE
C      THE MAIN PROGRAM ONLY DOES THE INTERACTIVE DIALOG TO
C      DETERMINE WHAT IS TO BE COMPUTED AND HOW ACCURATELY.
C
C      THE RANGE OF THE APPROXIMATION TO THE FUNCTION IS FROM
C      -AN TO AN.  IF THE USER SPECIFIES AN = 0 OR LESS THEN
C      A DEFAULT AN = 0 IS USED.
C
      PROGRAM TRIGOE(INPUT,OUTPUT)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AN, I
      EXTERNAL FSINT, SIN,
     X        FATANH, ATANH,
     X        FASINH, ASINH,
     X        FACOSH, ACOSH,
     X         FTANH, TANH,
     X         FASINT, ASIN,
     X          FTANT, TAN,
     X          FCOST, COS,
     X         FATANT, ATAN,
     X          FALNT, ALN,
C    X         FSHACT, SHAC,
     X         FSINH, SINH,
     X         FCOSH, COSH,
     X        FRASINT, RASIN
      DATA PI/3.14159 26535 [illegible] 784 26433 832/+D0
C
      WHILE (.TRUE.)
      .  PRINT *, ' ODD/EVEN FUNCTION   ',
      .  READ *, I
      .  PRINT 10, I
 10   .  FORMAT(/,A10)
      .  PRINT *, ' AN    ',
         READ *, AN
      .  PRINT *, AN
      .  CONDITIONAL
      .  .  (I .EQ. 1LHASINH)
      .  .  .  IF (AN [illegible] 0.0D0) AN = [illegible]0D0
      .  .  .  CALL POE(FASINH, ASINH)
      .  .  ...FIN
      .  .  (I .EQ. [illegible]ATANH)
      .  .  .  IF (AN [illegible] 0.0D0 [illegible]
      .  .  .  CALL [illegible]FATANH, [illegible]
      .  .  ...FIN
      .  .  (I .EQ. [illegible]TANH)
      .  .  .  IF (AN .LE. 0.0D0) AN = [illegible]
      .  .  .  CALL POE(FTANH, TANH)
      .  .  ...FIN
      .  .  (I .EQ. 1[illegible])
      .  .  .  IF (AN .LE. [illegible]) [illegible]
      .  .  .  CALL POE(F[illegible]
      .  .  ...FIN
```

```
      . . (I .EQ. 10HCOSH
      . . . IF (AN .LE. 0.0D0) AN = 0.5D0
      . . . CALL FOE(FCOSH), COSH
      . . ...FIN
C     . . (I .EQ. 10HSQAL
C     . . IF (AN .LE. 0.0D0) AN = 0.5D0
C     . . CALL FOE(FSQAL), SQAL
C     . . FIN
      . . (I .EQ. 10HSIN
      . . . IF (AN .LE. 0.0D0) AN = PI 4
      . . . CALL FOE(FSIN), SIN
      . . ...FIN
      . . (I .EQ. 10HASIN
      . . . IF (AN .LE. 0.0D0) AN = 0.375D0
      . . . CALL FOE(FASIN), ASIN
      . . ...FIN
      . . (I .EQ. 10HTAN
      . . . IF (AN .LE. 0.0D0) AN = PI 8
      . . . CALL FOE(FTAN), TAN
      . . ...FIN
      . . (I .EQ. 10HATAN
      . . . IF (AN .LE. 0.0D0) AN = DSQRT(2.0D0) - 1
      . . . CALL FOE(FATAN), ATAN
      . . ...FIN
      . . (I .EQ. 10HRASIN
      . . . IF (AN .LE. 0.0D0) AN = 0.5D0
      . . . CALL FOE(FRASIN), ASIN)
      . . ...FIN
      . . (I .EQ. 10H
      . . . IF (AN .LE. 0.0D0) AN = PI 4
      . . . CALL FOE
      . . ...FIN
      . . (I .EQ. 10H
      . . . IF (AN .LE. 0.0D0) AN
      . . . DSQRT(2.0D0) ... DSQRT(2.0D0) + 1)
      . . . CALL FOE(FASN), ASN
      . . ...FIN
      . . (OTHERWISE) PRINT *, ' I LL ASK AGAIN, '
      . ...FIN
      ...FIN
      END

      VERSION

      SUBROUTINE ... COMPUTE THE ODD OR EVEN CHEBYSHEV AND
      POLYNOMIAL ... FOR THE REQUESTED FUNCTION.

      ... THE FUNCTION ... (IE SIN, COS, ATAN
      ... AN, ... ) IS THE A ...
      BE INTEGRATED OVER THETA FROM 0 TO PI ... TO GIVE THE
      CHEBYSHEV COEFFICIENTS. ... AND N ARE PASSED THRU COMMON.

      ... IS THE MATRIX OF COEFFICIENTS FOR CONVERTING CHEBYSHEV
      COEFFICIENTS ... POLYNOMIAL COEFFICIENTS. THESE ARE
      COMPUTED IN ...

      ... IS THE ARRAY OF ... COEFFICIENTS OF THE
      FUNCTION ...
```

```
C
C       AS IS THE ARRAY OF COEFFICIENTS FOR THE POLYNOMIAL
C       APPROXIMATION OF SSSS, COMPUTED IN ACOEF.
C
C       THE ODD FUNCTIONS (SIN, TAN, ASIN, ATAN) ARE  CONVERTED
C       TO EVEN FUNCTIONS SIN(X)/X, TAN(X)/X, ASIN(X)/X, AND
C       ATAN(X)/X IN ORDER THAT THE RELATIVE ERROR BE UNIFORMLY
C       DISTRIBUTED BY THE CHEBYSHEV APPROXIMATION.
C
C       THE  ACCELERATED POLYNOMIAL APPROXIMATION IS VERIFIED
C       BY GENERATING AN ERROR CURVE OVER THE INTERVAL 0 TO AF.
C
      SUBROUTINE [illegible]
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON [illegible]
      DIMENSION [illegible]
      EQUIVALENCE [illegible]
C
      DIMENSION AS[illegible]
      EQUIVALENCE [illegible]
C
      DIMENSION [illegible]
      EQUIVALENCE [illegible]
C
      [illegible]
      INTEGER [illegible]
C
      [illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible] ***********
      [illegible] (43) 3327(1)
C
      [illegible]
      WR[illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible] SSSST, N [illegible]
      [illegible]
      [illegible]
      [illegible] FORMAT [illegible]
      [illegible]
C
      [illegible]
      [illegible]
      [illegible]
      PRINT [illegible] SUM
```

[illegible]

[illegible]

[illegible]

[illegible]

```
      CALL ACCEP(AS, C5555, IC, AN, N)                              00182
      N = N [illegible]                                             00183
      DO 99994 J=1,N                                                00184
      AS(J) = AS(2*J)                                               00184
99994 CONTINUE                                                      00184
      DES = DABS(1 - AS(1))                                         00186
      PRINT *, N, DES,                                              00187
      PRINT *, ' AN ', AN                                           00188
      DE = AMAX1(DES, DE)                                           00189
      MAX = 5                                                       00191
      DELTAX = AN [illegible] 249.                                  00192
      [illegible]                                                   00193
[illegible] GO TO 9999[illegible]                                   00194
      [illegible]                                                   00195
      DEMIN [illegible]                                             00196
      [illegible] MAX                                               00197
      [illegible] DELTA[illegible]                                  00198
      [illegible] THROUGH [illegible] GO TO 99989                   00200
      [illegible] ***, AS, N[illegible]                             00200
      [illegible]                                                   00201
      [illegible] GO TO 99988                                       00201
      [illegible] ***, AS, N                                        00201
      [illegible]                                                   00202
[illegible] GO TO 99987                                             00202
      [illegible] ***, AS, N[illegible]                             00202
      [illegible]                                                   00203
      [illegible] ***, AS, N[illegible]                             00204
      [illegible]                                                   00205
      DERRS = [illegible]5555 - E5555/E5555                         00206
      [illegible] DEMAX = DERRS                                     00207
      [illegible] DEMINS = DERRS                                    00208
[illegible]                                                         00209
      [illegible]*DEMIN[illegible]                                  00214
      [illegible]*DEMAX[illegible] DE = VS                          00215
      [illegible] + N[illegible]                                    00216
      [illegible] J=1,NS[illegible] J=1,NL), IS                     00217
      [illegible]                                                   00218
      [illegible]                                                   00219
[illegible] RETURN                                                  00220
      END                                                           00221
[illegible] 1 RECS. 144[illegible] WORDS.
```

COPY,OSUBOTH

```
C
C
C      PROGRAM BOTH CALCULATES CHEBYSHEV AND POLYNOMIAL APPROXIMATIONS
C      THAT USE BOTH ODD AND EVEN TERMS IN THEIR APPROXIMATIONS OF A
C      FUNCTION (EXP, LOG, SQRT).  ALL COMPUTATIONS ARE IN DOUBLE
C      PRECISION EXCEPT FOR THE ERROR TOLERANCE WHICH IS IN SINGLE.
C      THE ACTUAL COMPUTATIONS ARE CARRIED OUT IN THE SUBROUTINE PBOTH,
C      THE MAIN PROGRAM ONLY DOES THE INTERACTIVE DIALOG TO DETERMINE
C      WHAT IS TO BE COMPUTED AND HOW ACCURATELY.
C
C      THE RANGE OF THE APPROXIMATION TO THE FUNCTION IS FROM AL TO AN,
C      THERE ARE NO DEFAULT VALUES FOR AL OR AN.
C
      PROGRAM BOTH(INPUT,OUTPUT)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AN, I, AL
      COMMON/CONSTS/PI, ALN2
      EXTERNAL FLOGT, ALOG,
     X         FCBRTT, CBRT,
     X         FSQACT, SQAC,
     X          FTWOT, TWO,
     X          FEXPT, EXP,
     X         FEXIXT, EXIX,
     X         FAASNT, ASIN,
     X         FSQACT, SQAC,
     X         FSQRT, SQRT
      PI = 3.14159 26535 89793 23846 26433 832795D0
      ALN2 = DLOG(2.0D0)
C
      WHILE (.TRUE.)
      .  PRINT *, ' FUNCTION = ',
      .  READ *, I
      .  PRINT 10, I
10    .  FORMAT(X,A10)
      .  PRINT *, ' AN = ',
      .  READ *, AN
      .  PRINT *, AN
      .  PRINT *, ' AL = ',
      .  READ *, AL
      .  PRINT *, AL
      .  CONDITIONAL
      .  .  (I .EQ. 10HCBRT        ) CALL PBOTH(FCBRTT, CBRT)
      .  .  (I .EQ. 10HTWO         ) CALL PBOTH(FTWOT, TWO)
      .  .  (I .EQ. 10HEXIX        ) CALL PBOTH(FEXIXT, EXIX)
      .  .  (I .EQ. 10HLOG         ) CALL PBOTH(FLOGT, ALOG)
      .  .  (I .EQ. 10HAASIN       ) CALL PBOTH(FAASNT, ASIN)
      .  .  (I .EQ. 10HEXP         ) CALL PBOTH(FEXPT, EXP)
      .  .  (I .EQ. 10HSQRT        ) CALL PBOTH(FSQRT, SQRT)
      .  .  (I .EQ. 10HSQAC        ) CALL PBOTH(FSQACT, SQAC)
      .  .  (OTHERWISE) PRINT *, ' I'LL ASK AGAIN. ',
      .  ...FIN
      ...FIN
      END
```

(FLECS VERSION 22.51)

```
C
C     SUBROUTINE FROTH COMPUTES THE CHEBYSHEV AND POLYNOMIAL
C     COEFFICIENTS FOR THE REQUESTED FUNCTION.
C
C     SSSS IS THE FUNCTION TO BE APPROXIMATED (IE EXP, SQRT, ETC)
C     AND FSSSST IS  SSSS((AK+AL+(AK-AL)*COS(THETA))/2)*COS(N*THETA)
C     TO BE INTEGRATED OVER THETA FROM 0 TO PI TO GIVE THE
C     CHEBYSHEV COEFFICIENTS.  (AK, AL, AND N ARE AVAILABLE THRU
C     COMMON.)
C
C     IC IS THE MATRIX OF COEFFICIENTS FOR CONVERTING CHEBYSHEV
C     COEFFICIENTS TO POLYNOMIAL COEFFICIENTS,  THESE ARE CALCULATED
C     IN ICH.
C
C     CSSSS IS THE ARRAY OF CHEBYSHEV COEFFICIENTS FOR FUNCTION SSSS.
C
C     AS IS THE ARRAY OF COEFFICIENTS FOR THE POLYNOMIAL APPROXIMATION
C     OF SSSS,  COMPUTED IN ACOEF.
C
C     THE POLYNOMIAL APPROXIMATION IS VERIFIED BY GENERATING THE
C     RELATIVE ERROR CURVE OVER THE INTERVAL AL TO AK.
C
      SUBROUTINE FROTH(FSSSST, SSSS)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      COMMON N, AK, I, AL
      DIMENSION IC(25, 25), ICNEG(51)
      EQUIVALENCE (ICNEG(51), IC(1, 1))
C
      DIMENSION AS(25), AS0(2)
      EQUIVALENCE (AS0(2),AS(1))
C
      DIMENSION CSSSS0(3), CSSSS(25)
      EQUIVALENCE (CSSSS0(3),CSSSS(1))
C
      EXTERNAL SSSS, FSSSST
      INTEGER O
C
      DATA O/0/, ZERO/0.0D0/
      DATA CSSSS/25*0.0D0/
      DATA AS/25*0.0D0/
      DATA ICNEG/50*1000/, IC/625*1000/
      DATA IB/10H          /,IS/10H**********/
      DATA PI/3.14159 26535 89793 23846 26433 83279D0/
C
      PRINT *, ' QUAD ERR TOL ',
      READ *, OE
      PRINT *, OE
      PRINT *, ' FUNCT ERR TOL ',
      READ *, OEF
      PRINT *, OEF
C
      N = -1
      REPEAT WHILE (DABS(CSSSS(N)).GT.OEF)
      .  N = N + 1
      .  CSSSS(N) = 2*ROMBER(ZERO, PI, FSSSST, OE)/PI
      .  IF (N.EQ.0) CSSSS(N) = CSSSS(N)/2
      .  PRINT 5, N, CSSSS(N)
    5 .  FORMAT(I5, F35.25)
      ...FIN
C
```

```
      N = N - 1
      SUM = DABS(CSSSS(0))
      DO (J=1,N) SUM = SUM + DABS(CSSSS(J))
      PRINT *, N, ' SUMABS(SSSS) = ', SUM
      SUM = (CSSSS(0))
      DO (J=1,N) SUM = SUM + (CSSSS(J))
      PRINT *, N, ' SUM(SSSS) = ', SUM
      OEK = DABS(SSSS(AK) - SUM)
C
      CALL TCH(IC, N)
C
      XBAR = (AK + AL)/2
      CALL ACOEF(AS, CSSSS, IC, (AK - AL)/2, N)
C
      OES = DABS(SSSS(XBAR) - AS(0))
C
      OE = AMAX1(OES, OEK)
      PRINT *, N, OES, OEK,
      PRINT *, ' AK = ', AK,
      PRINT *, ' AL = ', AL
C
      JMAX = 5
      DELTAX = (AK - AL)/249.9
      X = AL
      WHILE (X .LT. AK)
      .  OEMAXS = 0
      .  OEMINS = 0
      .  DO (J=1,JMAX)
      .  .  X = X + DELTAX
      .  .  FSSSS = POLY(X - XBAR, AS, N)
      .  .  ESSSS = SSSS(X)
C     .  .  PRINT *, FSSSS, ESSSS
      .  .  OERRS = (FSSSS - ESSSS)/ESSSS
      .  .  IF (OERRS.GT.OEMAXS) OEMAXS = OERRS
      .  .  IF (OERRS.LT.OEMINS) OEMINS = OERRS
      .  ...FIN
C     .  PRINT *, OEMINS, OEMAXS, OEMINC, OEMAXC
C     .
C     .   GRAPHICAL OUT PUT OF ERROR CURVE
C     .
      .  NS = 30*OEMINS/OE
      .  NL = 30*OEMAXS/OE - NS
      .  NS = 65 + NS
      .  PRINT 10, (IB, J=1,NS), IS, (IB, J=1,NL), IS
 10   .  FORMAT(1X, 130A1)
      ...FIN
      RETURN
      END

   (FLECS VERSION 22.51)

   ------- ---------------------------------

C
      FUNCTION TWO(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON/CONSTS/PI, ALN2
      TWO = DEXP(X*ALN2)
C     PRINT*, ' X = ', X, ' TWO(X) = ', TWO
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION FTWOT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FTWOT = TWO((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION FCBRTT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON N, AK, I, AL
      EXTERNAL CBRT
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FCBRTT = CBRT((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION CBRT(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      CBRT = 0
      IF (X .EQ. 0) RETURN
      CBRT = DSIGN(DEXP(DLOG(DABS(X))/3), X)
C     PRINT*, ' X = ', X, ' CBRT(X) = ', CBRT
      RETURN
      END
```

(FLECS VERSION 22.51)

EOI. 0 FILES. 1 RECS. 893 WORDS.

```
COPY,COMPIL
      PROGRAM BOTH(INPUT,OUTPUT)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK, I, AL
      COMMON/CONSTS/PI, ALN2
      EXTERNAL FLOGT, ALOG,
     X        FCBRTT, CBRT,
     X        FSQACT, SQAC,
     X         FTWOT,  TWO,
     X         FEXPT,  EXP,
     X        FEX1XT, EX1X,
     X        FAASNT, ASIN,
     X        FSQACT, SQAC,
     X        FSQRTT, SQRT
      PI = 3.14159 26535 89793 23846 26433 83279D0
      ALN2 = DLOG(2.0D0)
99999 IF(.NOT.(.TRUE.)) GO TO 99998
      PRINT *, '  FUNCTION = ',
      READ *, I
      PRINT 10, I
10    FORMAT(X,A10)
      PRINT *, ' AK = ',
      READ *, AK
      PRINT *, AK
      PRINT *, ' AL =  ',
      READ *, AL
      PRINT *, AL
      IF(.NOT.(I .EQ. 10HCBRT       )) GO TO 99996
      CALL PBOTH(FCBRTT, CBRT)
      GO TO 99997
99996 IF(.NOT.(I .EQ. 10HTWO        )) GO TO 99995
      CALL PBOTH(FTWOT, TWO)
      GO TO 99997
99995 IF(.NOT.(I .EQ. 10HEX1X       )) GO TO 99994
      CALL PBOTH(FEX1XT, EX1X)
      GO TO 99997
99994 IF(.NOT.(I .EQ. 10HLOG        )) GO TO 99993
      CALL PBOTH(FLOGT, ALOG)
      GO TO 99997
99993 IF(.NOT.(I .EQ. 10HASASIN     )) GO TO 99992
      CALL PBOTH(FAASNT, ASIN)
      GO TO 99997
99992 IF(.NOT.(I .EQ. 10HEXP        )) GO TO 99991
      CALL PBOTH(FEXPT, EXP)
      GO TO 99997
99991 IF(.NOT.(I .EQ. 10HSQRT       )) GO TO 99990
      CALL PBOTH(FSQRTT, SQRT)
      GO TO 99997
99990 IF(.NOT.(I .EQ. 10HSQAC       )) GO TO 99989
      CALL PBOTH(FSQACT, SQAC)
      GO TO 99997
99989 PRINT *, ' I'LL ASK AGAIN, ',
99997 GO TO 99999
99998 END
      SUBROUTINE PBOTH(FSSSST, SSSS)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON N, AK, I, AL
      DIMENSION IC(25, 25), ICNEG(51)
      EQUIVALENCE (ICNEG(51), IC(1, 1))
      DIMENSION AS(25), AS0(2)
      EQUIVALENCE (AS0(2),AS(1))
```

```
      DIMENSION CSSSS0(3), CSSSS(25)
      EQUIVALENCE (CSSSS0(3),CSSSS(1))
      EXTERNAL SSSS, FSSSST
      INTEGER O
      DATA O/0/, ZERO/0.0D0/
      DATA CSSSS/25*0.0D0/
      DATA AS/25*0.0D0/
      DATA ICNEG/50*1000/, IC/625*1000/
      DATA IB/10H          /,IS/10H**********/
      DATA PI/3.14159 26535 89793 23846 26433 83279D0/
      PRINT *, ' QUAD ERR TOL ',
      READ *, OE
      PRINT *, OE
      PRINT *, ' FUNCT ERR TOL ',
      READ *, OEF
      PRINT *, OEF
      N = -1
      GO TO 99998
99999 IF(.NOT.(DABS(CSSSS(N)).GT.OEF)) GO TO 99997
99998 N = N + 1
      CSSSS(N) = 2*ROMBER(ZERO, PI, FSSSST, OE)/PI
      IF (N.EQ.0) CSSSS(N) = CSSSS(N)/2
      PRINT 5, N, CSSSS(N)
 5    FORMAT(I5, F35.25)
      GO TO 99999
99997 N = N - 1
      SUM = DABS(CSSSS(0))
      DO 99996 J=1,N
      SUM = SUM + DABS(CSSSS(J))
99996 CONTINUE
      PRINT *, N, ' SUMABS(SSSS) = ', SUM
      SUM = (CSSSS(0))
      DO 99995 J=1,N
      SUM = SUM + (CSSSS(J))
99995 CONTINUE
      PRINT *, N, ' SUM(SSSS) = ', SUM
      OEN = DABS(SSSS(AN) - SUM)
      CALL TCH(IC, N)
      XBAR = (AN + AL)/2
      CALL ACOEF(AS, CSSSS, IC, (AN - AL)/2, N)
      OES = DABS(SSSS(XBAR) - AS(0))
      OE = AMAX1(OES, OEN)
      PRINT *, N, OES, OEN,
      PRINT *, ' AN = ', AN,
      PRINT *, ' AL = ', AL
      JMAX = 5
      DELTAX = (AN - AL)/249.9
      X = AL
99994 IF(.NOT.(X .LT. AN)) GO TO 99993
      OEMAXS = 0
      OEMINS = 0
      DO 99992 J=1,JMAX
      X = X + DELTAX
      PSSSS = POLY(X - XBAR, AS, N)
      ESSSS = SSSS(X)
      OERRS = (PSSSS - ESSSS)/ESSSS
      IF (OERRS.GT.OEMAXS) OEMAXS = OERRS
      IF (OERRS.LT.OEMINS) OEMINS = OERRS
99992 CONTINUE
      NS = 30*OEMINS/OE
      NL = 30*OEMAXS/OE - NS
```

```
      NS = NS + NS
      PRINT 10, (IB, J=1,NS), IS, (IB, J=1,NL), IS
10    FORMAT(1X, 130A1)
      GO TO 99994
99994 RETURN
      END
      FUNCTION TWO(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON /CONSTS/PI, ALN2
      TWO = DEXP(X*ALN2)
      RETURN
      END
      FUNCTION FTWOT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON N, AN, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FTWOT = TWO((AN + AL + (AN - AL)*COST)/2)*COSNT
      RETURN
      END
      FUNCTION FCBRT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON N, AN, I, AL
      EXTERNAL CBRT
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FCBRT = CBRT((AN + AL + (AN - AL)*COST)/2)*COSNT
      RETURN
      END
      FUNCTION CBRT(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      CBRT = 0
      IF (X .EQ. 0) RETURN
      CBRT = DSIGN(DEXP(DLOG(DABS(X))/3), X)
      RETURN
      END
 EOI.   0 FILES.   1 RECS.   1413 WORDS.
```

/COPY,OSUBLIB

```
C
C
C      *     THIS FUNCTION DOES ROMBERG QUADRATURE ON THE FUNCTION F
C      *     FROM THE LOWER LIMIT A TO THE UPPER LIMIT B, AND TO THE
C      *     REQUESTED ACCURACY DE.
C      *     THIS  VERSION DOES NOT USE THE END POINTS.
C
       FUNCTION ROMBRG(A, B, F, DE)
       IMPLICIT DOUBLE PRECISION(A-H,P-Z)
C      DIMENSION DD(20)
       INTEGER FOUR(21), O
       DIMENSION Q(20), QQ(2)
       EQUIVALENCE (QQ(2),Q(1))
C
       DATA O/0/, LMAX/1/, FOUR(1)/4/
C
       Q(O) = 0
       H = (B - A)/2
       IMAX = 1
C
       DO (L=1,20)
       .  IF (L.GE.LMAX)
       .  .  LMAX = L + 1
       .  .  FOUR(L+1) = 4*FOUR(L)
       .  ...FIN
C      .
       .  SUM = 0
       .  DO (I=1,IMAX)
       .  .  V = (2*I - IMAX - 1.0D0)/IMAX
       .  .  U = V*(3 - V*V)/2
       .  .  UP = 1.5D0*(1 - V*V)
       .  .  X = A + (U + 1)*H
       .  .  SUM = SUM + F(X)*UP
       .  ...FIN
       .  SUM = SUM + IMAX*Q(L-1)
       .  IMAX = 2*IMAX
       .  Q(L) = SUM/IMAX
C      .
       .  DO(M=1,L)
       .  .  K = L - M
       .  .  Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1)
       .  ...FIN
C      .
       .  IF (L.GT.1)
       .  .  ROMBRG = (B - A)*Q(0)
C      .  .
C      .  .  DO (K-1,L) DD(K) = (Q(0) - Q(K))*(B - A)
C      .  .  PRINT*, ROMBRG, (DD(K), K=1,L)
C      .  .
       .  .  ODQ = DABS((Q(0) - Q(1))*(B - A))
       .  .  IF (ODQ.LE.DE) RETURN
       .  ...FIN
C      .
       ...FIN
       RETURN
```

```
      END
```

(FLECS VERSION 22.51)

```
C
C
C     *     THIS FUNCTION DOES ROMBERG QUADRATURE ON THE FUNCTION F
C     *     FROM THE LOWER LIMIT A TO THE UPPER LIMIT B, AND TO THE
C     *     REQUESTED ACCURACY OE.
C
      FUNCTION ROMBER(A, B, F, OE)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
C     DIMENSION OD(20)
      INTEGER FOUR(21), O
      DIMENSION Q(20), QO(2)
      EQUIVALENCE (QO(2),Q(1))
C
      DATA O/O/, LMAX/1/, FOUR(1)/4/
C
      Q(O) = (F(A) + F(B))/2
      H = B - A
      IMAX = 1
C
      DO (L=1,20)
      .  IF (L.GE.LMAX)
      .  .  LMAX = L + 1
      .  .  FOUR(L+1) = 4*FOUR(L)
      .  ...FIN
C     .
      .  H = H/2
      .  SUM = 0
      .  DO (I=1,IMAX) SUM = SUM + F(A + H*(2*I - 1))
      .  SUM = SUM + IMAX*Q(L-1)
      .  IMAX = 2*IMAX
      .  Q(L) = SUM/IMAX
C     .
      .  DO(M=1,L)
      .  .  K = L - M
      .  .  Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1)
      .  ...FIN
C     .
      .  ROMBER = (B - A)*Q(O)
C     .
C     .  DO (K=1,L) OD(K) = (Q(O) - Q(K))*(B - A)
C     .  PRINT*, ROMBER, (OD(K), K=1,L)
C     .
      .  ODQ = DABS((Q(O) - Q(1))*(B - A))
      .  IF (ODQ.LE.OE) RETURN
C     .
      ...FIN
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
C
```

```
      FUNCTION SIN(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      SIN = DSIN(X)
C     PRINT*, ' X =', X, ' SIN(X) =', SIN
      RETURN
      END

   (FLECS VERSION 22.51)

   ----------------------------------------

C
      FUNCTION TAN(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      TAN = DSIN(X)/DCOS(X)
C     PRINT*, ' X =', X, ' TAN(X) =', TAN
      RETURN
      END

   (FLECS VERSION 22.51)

   ----------------------------------------

C
      FUNCTION ASIN(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
       ASIN = DATAN(X/DSQRT(1-X*X))
C     PRINT*, ' X =', X, ' ASIN(X) =', ASIN
      RETURN
      END

   (FLECS VERSION 22.51)

   ----------------------------------------

C
      FUNCTION ACOS(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ACOS = DATAN(DSQRT(1-X*X)/X)
C     PRINT*, ' X =', X, ' ACOS(X) =', ACOS
      RETURN
      END

   (FLECS VERSION 22.51)

   ----------------------------------------

C
      FUNCTION EXP(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXP = DEXP(X)
C     PRINT*, ' X =', X, ' EXP(X) =',  EXP
      RETURN
      END

   (FLECS VERSION 22.51)

   ----------------------------------------

C
      FUNCTION SQRT(X)
```

```
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      SQRT = DSQRT(X)
C     PRINT*, ' X =', X, ' SQRT(X) =', SQRT
      RETURN
      END

 (FLECS VERSION 22.51)


C
      FUNCTION ATAN(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ATAN = DATAN(X)
C     PRINT*, ' X =', X, ' ATAN(X) =', ATAN
      RETURN
      END

 (FLECS VERSION 22.51)


C
      FUNCTION ALOG(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ALOG = DLOG(X)
      PRINT*, ' X =', X, ' ALOG(X) =', ALOG
      RETURN
      END

 (FLECS VERSION 22.51)


C
      FUNCTION COS(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COS = DCOS(X)
C     PRINT*, ' X =', X, ' COS(X) =', COS
      RETURN
      END

 (FLECS VERSION 22.51)


C
      FUNCTION FCOST(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AN
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FCOST = DCOS(AN*COST)*COSNT
      RETURN
      END

 (FLECS VERSION 22.51)


C
```

```
      FUNCTION FSINT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FSINT = COSNT
      IF (COST.EQ.0) RETURN
      FSINT = (DSIN(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END
```

FLECS VERSION [illegible]

```
C
C
      FUNCTION PROD(A, N)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      DIMENSION A(*)
      PROD = A(N)
      DO (J=1,N) PROD = PROD* + A(N-J)
      PROD = PROD
      RETURN
      END
```

FLECS VERSION [illegible]

```
C
      SUBROUTINE [illegible] E, IC, AK, [illegible]
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      DIMENSION [illegible] (25), IC [illegible] 25)
      N = 4
      DO (I=1,N)
      . SUM = [illegible]
      . DO (J=1,N) SUM = SUM + E [illegible] *IC(N, J)
      . A [illegible] = SUM [illegible] **
C     . PRINT *, [illegible] A
      [illegible]
      [illegible]
      [illegible]
      SUM = [illegible]
      DO (J=1,N) SUM = SUM + [illegible]
      [illegible] = SUM
C     PRINT *, [illegible] A [illegible]
      RETURN
      END
```

FLECS VERSION [illegible]

```
C
      SUBROUTINE TCH(IC, N)
      DIMENSION IC(25, 25)
      INTEGER U
      DATA U(0)
C
      IC(0, 0) = 1
```

```
      [illegible] 1, [illegible]
      DO [illegible] 1,N
      .     [illegible] [illegible] 2, [illegible]))
      .  [illegible] 2, [illegible]
      .  [illegible] 1, 1 [illegible]
      .  [illegible]
      .  .  WHEN (MOD([illegible]).EQ.1) IC(K, J) = 0
      .  .  ELSE IC([illegible]) [illegible]*IC(K-1, J-1) - IC(K-2, J)
      .     PRINT *, [illegible]
      .  ...FIN
      ...FIN
      RETURN
      END

[illegible] VERSION [illegible]

      FUNCTION FTAN (THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL TAN
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FTANT = COSNT
      IF (COST.EQ.0) RETURN
      FTANT = (TAN(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END

[illegible] VERSION [illegible]

      FUNCTION FATANT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FATANT = COSNT
      IF (COST.EQ.0) RETURN
      FATANT = (DATAN(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END

FLE[illegible] VERSION [illegible]

      FUNCTION FASINT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ASIN
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FASINT = COSNT
      IF (COST.EQ.0) RETURN
      FASINT = (ASIN(AK*COST)/AK)*(COSNT/COST)
```

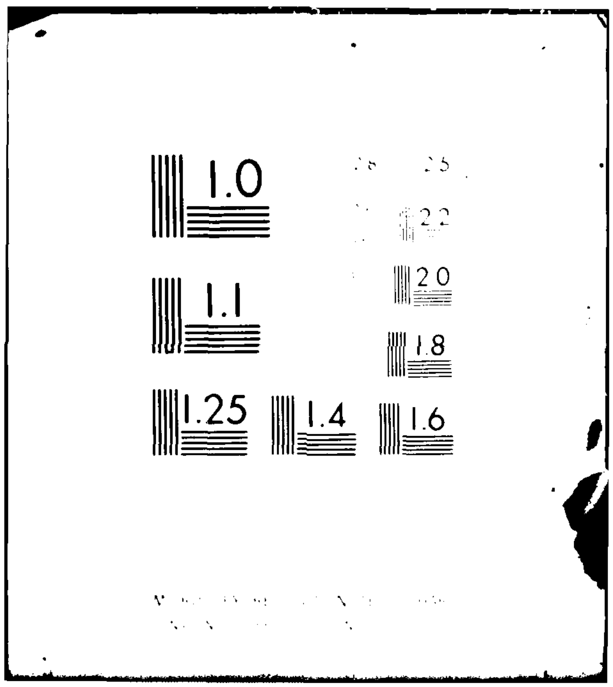
AD-A116 070 GEORGIA INST OF TECH ATLANTA ENGINEERING EXPERIMENT --ETC F/G 9/2
INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE--ETC(U)
MAR 82 L J GALLAHER F30602-78-C-0120

UNCLASSIFIED RADC-TR-82-46 NL

2 OF 2

AD A
116070

END
DATE
FILMED
07-82
DTIC


1.0
2.2
2.0
1.1
1.8
1.25
1.4
1.6

```
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION FACOST(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ACOS
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FACOST = ACOS(AK*COST)*COSNT
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION RASIN(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ASIN
      RASIN = 1/ASIN(X)
C     PRINT*, ' X =', X, ' RASIN(X) =', RASIN
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION FRASINT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL RASIN
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FRASINT = COSNT
      IF  (COST.EQ.0) RETURN
      FRASINT = COST*RASIN(AK*COST)*AK*COSNT
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION FALNT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ALN
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FALNT = COSNT
      IF  (COST.EQ.0) RETURN
```

```
      FALNT = (ALN(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END

    (FLECS VERSION 22.51)

    --------------------------------------

C
      FUNCTION ALN(Z)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ALN = DLOG((Z + 1)/(1 - Z))/2
C     PRINT*, ' Z =', Z, ' ALN(Z) =',  ALN
      RETURN
      END

    (FLECS VERSION 22.51)

    --------------------------------------

C
      FUNCTION FLOGT(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FLOGT = DLOG((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END

    (FLECS VERSION 22.51)

    --------------------------------------

C
      FUNCTION FSQRTT(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FSQRTT = DSQRT((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END

    (FLECS VERSION 22.51)

    --------------------------------------

C
      FUNCTION FEXPT(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FEXPT = DEXP((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END

    (FLECS VERSION 22.51)

    --------------------------------------
```

```
C
      FUNCTION FAASNT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ASIN
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FAASNT = ASIN((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION EX1X(X)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      EX1X = 1
      IF (X.NE.0) EX1X = (DEXP(X) - 1)/X
C     PRINT*, ' X =', X, ' EX1X(X) =', EX1X
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION FEX1XT(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FEX1XT = EX1X((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
      FUNCTION SQAC(Y)
C     NOTE . .  THIS FUNCTION COMPUTES 1/2 OF ACOS(1-Y)**2/Y
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
C     EXTERNAL ACOS
      SQAC = 1
      IF (Y.EQ.0) RETURN
C     SQAC = ACOS(1 - Y)**2/(2*Y)
      SQAC = DATAN(DSQRT(Y*(2 - Y))/(1 - Y))**2/(2*Y)
C     PRINT*, 'Y = ', Y, '  SQAC(Y) = ', SQAC
      RETURN
      END
```

(FLECS VERSION 22.51)

```
C
```

```
      FUNCTION FSQACT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FSQACT = SQAC((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END

  (FLECS VERSION 22.51)

  ----------------------------------------

C
      FUNCTION SINH(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      E = DEXP(X)
      SINH = (E - 1/E)/2
C     PRINT*, ' X = ', X, '  SINH(X) = ', SINH
      RETURN
      END

  (FLECS VERSION 22.51)

  ----------------------------------------

C
      FUNCTION  COSH(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      E = DEXP(X)
      COSH = (E + 1/E)/2
C     PRINT*, ' X = ', X, '  COSH(X) = ', COSH
      RETURN
      END

  (FLECS VERSION 22.51)

  ----------------------------------------

C
      FUNCTION FSINHT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL SINH
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FSINHT = COSNT
      IF (COST.EQ.0) RETURN
      FSINHT = (SINH(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END

  (FLECS VERSION 22.51)

  ----------------------------------------

C
      FUNCTION FCOSHT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL COSH
      COMMON N, AK
```

```
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FCOSHT = COSH(AK*COST)*COSNT
      RETURN
      END

  (FLECS VERSION 22.51)

  ----------------------------------------

C
      FUNCTION ASINH(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ATANH
      ASINH = 0.0D0
      IF (X .EQ. 0) RETURN
      ASINH = ATANH(X/DSQRT(1 + X*X))
C     PRINT*, ' X =', X, ' ASINH(X) =', ASINH
      RETURN
      END

  (FLECS VERSION 22.51)

  ----------------------------------------

C
      FUNCTION FASINHT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ASINH
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FASINHT = COSNT
      IF (COST.EQ.0) RETURN
      FASINHT = (ASINH(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END

  (FLECS VERSION 22.51)

  ----------------------------------------

C
      FUNCTION ACOSH(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ATANH
      ACOSH = ATANH(DSQRT(X*X-1)/X)
C     PRINT*, ' X =', X, ' ACOSH(X) =', ACOSH
      RETURN
      END

  (FLECS VERSION 22.51)

  ----------------------------------------

C
      FUNCTION FACOSHT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ACOSH
      COMMON N, AK
      COST = DCOS(THETA)
```

```
      COSNT = DCOS(N*THETA)
      FACOSHT = ACOSH(AK*COST)*COSNT
      RETURN
      END

   (FLECS VERSION 22.51)

   ------------------------------------------

C
      FUNCTION ATANH(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ATANH = 0.0D0
      IF (X .EQ. 0) RETURN
      ATANH = DLOG((1+X)/(1-X))/2
C     PRINT*, ' X =', X, ' ATANH(X) =', ATANH
      RETURN
      END

   (FLECS VERSION 22.51)

   ------------------------------------------

C
      FUNCTION FATANHT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ATANH
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FATANHT = COSNT
      IF  (COST.EQ.0) RETURN
      FATANHT = (ATANH(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END

   (FLECS VERSION 22.51)

   ------------------------------------------

C
      FUNCTION TANH(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      TANH = DTANH(X)
C     PRINT*, ' X =', X, ' TANH(X) =', TANH
      RETURN
      END

   (FLECS VERSION 22.51)

   ------------------------------------------

C
      FUNCTION FTANHT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL TANH
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FTANHT = COSNT
      IF  (COST.EQ.0) RETURN
```

```
      FTANHT = (TANH(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END

   (FLECS VERSION 22.51)

   ----------------------------------------

EOI.   0 FILES.   1 RECS.   2321 WORDS.
/
```

```
COPY,COMPIL
      FUNCTION ROMBRG(A, B, F, OE)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      INTEGER FOUR(21), O
      DIMENSION Q(20), QO(2)
      EQUIVALENCE (QO(2),Q(1))
      DATA O/0/, LMAX/1/, FOUR(1)/4/
      Q(O) = 0
      H = (B - A)/2
      IMAX = 1
      DO 99999 L=1,20
      IF(.NOT.(L.GE.LMAX)) GO TO 99998
      LMAX = L + 1
      FOUR(L+1) = 4*FOUR(L)
99998 SUM = 0
      DO 99997 I=1,IMAX
      V = (2*I - IMAX - 1.0D0)/IMAX
      U = V*(3 - V*V)/2
      UP = 1.5D0*(1 - V*V)
      X = A + (U + 1)*H
      SUM = SUM + F(X)*UP
99997 CONTINUE
      SUM = SUM + IMAX*Q(L-1)
      IMAX = 2*IMAX
      Q(L) = SUM/IMAX
      DO 99996 M=1,L
      K = L - M
      Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1)
99996 CONTINUE
      IF(.NOT.(L.GT.1)) GO TO 99995
      ROMBRG = (B - A)*Q(O)
      ODQ = DABS((Q(O) - Q(1))*(B - A))
      IF (ODQ.LE.OE) RETURN
99995 CONTINUE
99999 CONTINUE
      RETURN
      END
      FUNCTION ROMBER(A, B, F, OE)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      INTEGER FOUR(21), O
      DIMENSION Q(20), QO(2)
      EQUIVALENCE (QO(2),Q(1))
      DATA O/0/, LMAX/1/, FOUR(1)/4/
      Q(O) = (F(A) + F(B))/2
      H = B - A
      IMAX = 1
      DO 99999 L=1,20
      IF(.NOT.(L.GE.LMAX)) GO TO 99998
      LMAX = L + 1
      FOUR(L+1) = 4*FOUR(L)
99998 H = H/2
      SUM = 0
      DO 99997 I=1,IMAX
      SUM = SUM + F(A + H*(2*I - 1))
99997 CONTINUE
      SUM = SUM + IMAX*Q(L-1)
      IMAX = 2*IMAX
      Q(L) = SUM/IMAX
      DO 99996 M=1,L
      K = L - M
      Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1)
```

```
99996 CONTINUE
      ROMBER = (B - A)*Q(0)
      ODQ = DABS((Q(0) - Q(1))*(B - A))
      IF (ODQ.LE.OE) RETURN
99999 CONTINUE
      RETURN
      END
      FUNCTION SIN(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      SIN = DSIN(X)
      RETURN
      END
      FUNCTION TAN(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      TAN = DSIN(X)/DCOS(X)
      RETURN
      END
      FUNCTION ASIN(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
       ASIN = DATAN(X/DSQRT(1-X*X))
      RETURN
      END
      FUNCTION ACOS(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ACOS = DATAN(DSQRT(1-X*X)/X)
      RETURN
      END
      FUNCTION EXP(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXP = DEXP(X)
      RETURN
      END
      FUNCTION SQRT(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      SQRT = DSQRT(X)
      RETURN
      END
      FUNCTION ATAN(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ATAN = DATAN(X)
      RETURN
      END
      FUNCTION ALOG(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ALOG = DLOG(X)
      RETURN
      END
      FUNCTION COS(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COS = DCOS(X)
      RETURN
      END
      FUNCTION FCOST(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FCOST = DCOS(AK*COST)*COSNT
      RETURN
      END
      FUNCTION FSINT(THETA)
```

```
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      COMMON N, AK
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FSINT = COSNT
      IF  (COST.EQ.0) RETURN
      FSINT = (DSIN(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END
      FUNCTION POLY(X, A, N)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      DIMENSION A(25)
      PROD = A(N)
      DO 99999 J=1,N
      PROD = PROD*X + A(N-J)
99999 CONTINUE
      POLY = PROD
      RETURN
      END
      SUBROUTINE ACOEF(A, B, IC, AK, N)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      DIMENSION B(25), A(25), IC(25,25)
      DO 99999 J=1,N
      SUM = 0
      DO 99998 K=J,N
      SUM =SUM + B(K)*IC(K, J)
99998 CONTINUE
      A(J) = SUM/AK**J
99999 CONTINUE
      J = 0
      K = 0
      SUM = B(K)*IC(K, J)
      DO 99997 K=1,N
      SUM = SUM + B(K)*IC(K, J)
99997 CONTINUE
      A(J) = SUM
      RETURN
      END
      SUBROUTINE TCH(IC, N)
      DIMENSION IC(25, 25)
      INTEGER O
      DATA O/0/
      IC(O, O) = 1
      IC(O-1, O) = 0
      DO 99999 K=1,N
      IC(K, O) = -IC(K-2, O)
      IC(K-2, K) = 0
      IC(O-1, 1) = 1
      DO 99998 J=1,K
      IF(.NOT.(MOD(J+K,2).EQ.1)) GO TO 99996
      IC(K, J) = 0
      GO TO 99997
99996 IC(K, J) = 2*IC(K-1, J-1) - IC(K-2, J)
99997 CONTINUE
99998 CONTINUE
99999 CONTINUE
      RETURN
      END
      FUNCTION FTANT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
       XTERNAL
```

```
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FTANT = COSNT
IF  (COST.EQ.0) RETURN
FTANT = (TAN(AK*COST)/AK)*(COSNT/COST)
RETURN
END
FUNCTION FATANT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FATANT = COSNT
IF  (COST.EQ.0) RETURN
FATANT = (DATAN(AK*COST)/AK)*(COSNT/COST)
RETURN
END
FUNCTION FASINT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ASIN
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FASINT = COSNT
IF  (COST.EQ.0) RETURN
FASINT = (ASIN(AK*COST)/AK)*(COSNT/COST)
RETURN
END
FUNCTION FACOST(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ACOS
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FACOST = ACOS(AK*COST)*COSNT
RETURN
END
FUNCTION RASIN(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ASIN
RASIN = 1/ASIN(X)
RETURN
END
FUNCTION FRASINT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL RASIN
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FRASINT = COSNT
IF  (COST.EQ.0) RETURN
FRASINT = COST*RASIN(AK*COST)*AK*COSNT
RETURN
END
FUNCTION FALNT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ALN
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
```

```
      FALNT = COSNT
      IF  (COST.EQ.0) RETURN
      FALNT = (ALN(AK*COST)/AK)*(COSNT/COST)
      RETURN
      END
      FUNCTION ALN(Z)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ALN = DLOG((Z + 1)/(1 - Z))/2
      RETURN
      END
      FUNCTION FLOGT(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FLOGT = DLOG((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END
      FUNCTION FSQRTT(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FSQRTT = DSQRT((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END
      FUNCTION FEXPT(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FEXPT = DEXP((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END
      FUNCTION FAASNT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ASIN
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FAASNT = ASIN((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END
      FUNCTION EX1X(X)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      EX1X = 1
      IF (X.NE.0) EX1X = (DEXP(X) - 1)/X
      RETURN
      END
      FUNCTION FEX1XT(THETA)
      IMPLICIT DOUBLE PRECISION (A-H,P-Z)
      COMMON N, AK, I, AL
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FEX1XT = EX1X((AK + AL + (AK - AL)*COST)/2)*COSNT
      RETURN
      END
      FUNCTION SQAC(Y)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      SQAC = 1
      IF (Y.EQ.0) RETURN
```

```
SQAC = DATAN(DSQRT(Y*(2 - Y))/(1 - Y))**2/(2*Y)
RETURN
END
FUNCTION FSQACT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FSQACT = SQAC((AK + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END
FUNCTION SINH(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
E = DEXP(X)
SINH = (E - 1/E)/2
RETURN
END
FUNCTION  COSH(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
E = DEXP(X)
COSH = (E + 1/E)/2
RETURN
END
FUNCTION FSINHT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL SINH
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FSINHT = COSNT
IF (COST.EQ.0) RETURN
FSINHT = (SINH(AK*COST)/AK)*(COSNT/COST)
RETURN
END
FUNCTION FCOSHT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL COSH
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FCOSHT = COSH(AK*COST)*COSNT
RETURN
END
FUNCTION ASINH(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ATANH
ASINH = 0.0D0
IF (X .EQ. 0) RETURN
ASINH = ATANH(X/DSQRT(1 + X*X))
RETURN
END
FUNCTION FASINHT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ASINH
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FASINHT = COSNT
IF  (COST.EQ.0) RETURN
FASINHT = (ASINH(AK*COST)/AK)*(COSNT/COST)
RETURN
```

```
      END
      FUNCTION ACOSH(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ATANH
      ACOSH = ATANH(DSQRT(X*X-1)/X)
      RETURN
      END
      FUNCTION FACOSHT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ACOSH
      COMMON N, AN
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FACOSHT = ACOSH(AN*COST)*COSNT
      RETURN
      END
      FUNCTION ATANH(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      ATANH = 0.0D0
      IF (X .EQ. 0) RETURN
      ATANH = DLOG((1+X)/(1-X))/2
      RETURN
      END
      FUNCTION FATANHT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL ATANH
      COMMON N, AN
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FATANHT = COSNT
      IF (COST.EQ.0) RETURN
      FATANHT = (ATANH(AN*COST)/AN)*(COSNT/COST)
      RETURN
      END
      FUNCTION TANH(X)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      TANH = DTANH(X)
      RETURN
      END
      FUNCTION FTANHT(THETA)
      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
      EXTERNAL TANH
      COMMON N, AN
      COST = DCOS(THETA)
      COSNT = DCOS(N*THETA)
      FTANHT = COSNT
      IF (COST.EQ.0) RETURN
      FTANHT = (TANH(AN*COST)/AN)*(COSNT/COST)
      RETURN
      END
EDT   [illegible],    1 RECS.    3735 WORDS.
```

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

FILMED

7-8